MODELLING COOPERATING MULTI-MANIPULATOR WORKCELLS

By

SUBBIAN GOVINDARAJ

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN
PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

1988

To My Parents

for all the love
they have given me

ACKNOWLEDGEMENTS

I shall forever be deeply indebted to Dr. Keith L. Doty for educating me and stimulating my creative thinking. I thank him for the countless hours he spent discussing, debating, directing, reading and correcting my work. He taught me to be a better student, teacher, manager and thinker.

I thank my entire committee, Dr. Robert Sullivan, Dr. Herman Lam, Dr. Giuseppe Basile and Dr. Randy Chow, for taking time out of their busy schedules to read and correct my dissertation to make it a better piece of work.

I am very grateful to my parents and family for being patient and supportive. Theirs was the dream I had the good fortune of fulfilling. I give them my deepest love and thanks. In particular, I can never repay Dr. Asokan, Sagi and Anupa for giving me a home away from home.

My love, thanks and apologies go to my wife Nalini for being supportive during the writing of this dissertation and putting up with all the associated madness.

I would like to thank my numerous colleagues, past and present, for all the insights they gave me and the most pleasant working atmosphere. With them around it was more fun than work. In particular, I thank Sujeet for planting the seed for this dissertation, Eric Shwartz and Sumant Pal for reading and debating it, David Alderman for the millions of things he does without which the Machine Intelligence Lab cannot be as productive and all the members of the Machine Intelligence Laboratory for providing a sounding board for my ideas.

My thanks go to Dr. Stanley Su and the staff of his Database Systems Research and Development Center, University of Florida, for their inputs about OSAM* and various discussions about MEDL and OSAM*.

I acknowledge with gratitude the support provided by the Electrical Engineering Department of the University of Florida without which this work could never have been completed.

My thanks go to my brother Thiagaraj and my many friends Chandra, Louiqa, Geeta, Prem, Shankar, Malek, . . . who make life so enjoyable, just knowing them and sharing it with them.

And finally the Gators. They made my long stay in Gainesville pleasant, happy and unforgetable.

TABLE OF CONTENTS

Abstract of Dissertation Presented to the Graduate School of the University of Florida in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

MODELLING COOPERATING MULTI-MANIPULATOR WORKCELLS

By

SUBBIAN GOVINDARAJ

April 1988

Chairman: Dr. Keith L. Doty
Major Department: Electrical Engineering

Many separate computer software tools are available for manufacturing engineers to take a product through its life-cycle of design, simulation and production in manufacturing workcells. A manufacturing engineer is forced to possess a considerable level of expertise in computing systems and languages to use these various programs. This dissertation develops manufacturing workcell object and task modelling schema that can be used to integrate manufacturing software. The modelling schema developed here helps to support various underlying application views while preserving a user's high level view of the workcell models and tasks.

A high-level, computer-language-independent, object model description formalism called "Manufacturing Engineers' Description Language" (MEDL) is presented. This language permits description of manufacturing workcell "world

models." MEDL supports five functional basis elements--statica, informata, sensors, actuators and processes--and provides a limited number of operations on this basis. Workcell objects and configurations are modelled using "aggregations" of MEDL basis elements and operations.

Task modelling primitives developed in this dissertation support single and multi-manipulator motion tasks. Moveto, attach and detach are identified as the basic single-robot motion commands. Multi-manipulator tasks are partitioned into three periods: rendezvous, pre-rendezvous and post-rendezvous. During the critical rendezvous period, two types of cooperation are possible between manipulators. Static manipulation where the workpiece is stationary during rendezvous and Kinetic manipulation where the workpiece is in motion are modelled. The tasking primitives can be used by a general purpose task planner to plan high-level, user task requests for multi-manipulator workcells.

CHAPTER ONE

INTRODUCTION

Robotics research conducted throughout the world has made many advances in the past decade. These advances have spurned a lot of interest in robotics and speculation that robotics technology and Computer Aided Manufacturing (CAM) were going to provide U.S. industries the ability to regain their supremacy in the manufacture of goods. To date, robotics technology has not lived up to this expectation, resulting in disillusionment about the role of robotics in the economic future of the U.S.

Basic research in robotics still proceeds in different areas: mechanisms, control, sensors, controllers and languages. CAM and Computer Integrated Manufacturing (CIM) technologies help tie together robots and other "intelligent" manufacturing devices into workcells and plant-floors. Predictions for the "factories of the future" paint the scenario of computer networks connecting together multiple, computer controlled workcells into manufacturing plant-floors and plants. Automated planning for product manufacturing with hardly any human intervention is also proposed. The National Bureau of Standards (NBS) has set up a prototype facility called the Automated Manufacturing

Research Facility (AMRF) for studying such computer integrated factories of the future [Simpson et al. 1982]. AMRF helps study the planning and control aspects of a multi-level control hierarchy in manufacturing facilities. Westinghouse's TROIKABOT project investigates the application of NBS's control hierarchy for a multi-manipulator workcell [Acker et al. 1985].

Robots are integral parts of manufacturing workcells with responsibilities including parts delivery, machining, assembly, and inspection just to name a few. Even today, in automobile assembly lines, one finds multiple robot manipulators sharing work areas in manufacturing workcells. In order to efficiently utilize the actions of multiple manipulators, studies about robot cooperation are inevitable.

Cooperating multiple-manipulators provide many advantages [Doty 1983, Acker et al. 1985]. One of the advantages pointed out is improved task cycle times. Tasks are made more efficient by cooperating robots signaling each other instead of waiting on a different mechanism, usually sensors, for synchronization. Another advantage is load sharing between cooperating manipulators. Load sharing results in reduced payload capacity robots being sufficient for the tasks. Reduced payload robots are usually less expensive and more dexterous. In some examples, like lifting a long bar, cooperating robots may be the only way of achieving the task.

Recently, there has been increased interest in multi-robot cooperation for object manipulation. Research has focused on the issues of load sharing between multiple robots connected together through a manipulated workpiece [Chand 1984, Nakamura et al. 1987, Luh and Zheng 1987]. The issues being investigated here are the force interactions, compliance and control during cooperative manipulation of objects. Controllers for cooperative manipulation are being designed at the servo and command levels of the control hierarchy [Zheng et al. 1987, Tarn et al. 1986, Stauffer 1980].

Current research into cooperating robots focuses on a narrow portion of a cooperating task--only the time when both participating manipulators interact with the workpieces. In order for a cooperating task to be successful, the periods of the task before and after the cooperation period are also critical. The current techniques for specifying tasks for single robots involves teach-pendant based programming. This technique will fail for cooperation tasks due to the multiple robots involved and synchronizations between them. Their programming, therefore, invariably involves the manipulated workpiece during cooperational task specification.

For manufacturing facilities, in general, off-line task verification is becoming a valuable tool. Experimental and commercial facilities are available for 3-D graphical simulation of tasks in manufacturing workcells [Hornick and

Ravani 1986, Mogal 1986, Crane 1987]. Simulation tools become invaluable for the complex tasks of multi-manipulator cooperation. Such tasks have to be successfully simulated before actual testing in workcells.

AMRF, TROIKABOT and the existing simulation systems demonstrate a need for large databases of manufacturing "world model" data. Database researchers are addressing different modelling techniques for maintaining such large databases [Kemper and Wallrath 1987, Su 1986]. Su et al. [1988] focuses on applying an object-oriented data modelling schema (OSAM*) for maintaining manufacturing databases using the AMRF databases as an example. All the data maintenance and application systems assume the availability of the large databases. Usually these databases are manually created and entered into the systems and have no mechanisms for maintaining the data integrity of the models.

There are no standard world modelling rules available for workcell models maintained in manufacturing databases and used by application software. Each tool is built with its own model representational scheme and application languages. A manufacturing database user, therefore, has to have a fairly high level of computer expertise to understand modelling in the various systems and languages.

This dissertation focuses on defining a high level modelling framework for describing world models for manufacturing workcells and high-level, language independent tasking primitives for defining multi-robot cooperation tasks. The dissertation also describes a knowledge generation scheme for deriving detailed device knowledge from high-level descriptions of manufacturing devices.

In Chapter Three, we develop an object modelling methodology, called a Manufacturing Engineers' Description Language (MEDL), which gives a high-level world model description scheme. MEDL gives manufacturing engineers a tool for describing workcell devices and workcell models. A MEDL Interpreter (MEDLI) can provide a front-end for manufacturing database modelling and application facilities to convert a manufacturing engineer's view of the workcells into the representational scheme of the target systems.

Automated task planners are invaluable for taking workpiece oriented task specifications, splitting them into task requests for the cooperating robots and generating the required high-level synchronizations and data interactions between the participating manipulators. In Chapter Four of this dissertation, we develop robot independent tasking primitives for multi-robot cooperational tasks. A recent NSF workshop on cooperating robots expressed a need for developing such primitives [NSF 1987]. The Machine Intelligence Laboratory (MIL) is also investigating a Task Planner to generate plan-trees for multi-manipulator cooperation tasks using the tasking primitives developed here [Pal and Doty 1987].

Currently, generation of manufacturing databases is a manual operation. This slows down the generation of data and is not guaranteed to be error-free. In Chapter Five, we describe a knowledge derivation scheme for generating detailed knowledge about manufacturing devices from their high-level MEDL specifications. This scheme can be used for automated, error-free knowledge generation for manufacturing databases. We use a symbolic knowledge generator for manipulators (kinema objects) to illustrate the concept. The symbolic data can be converted to any target application representation scheme.

The modelling tools and primitives developed in this dissertation have applications in current technology tools. Together they provide the tools necessary for integrating various manufacturing application software into providing a total facility for modelling manufacturing tasks from product design, to simulation and eventually to the actual production in a manufacturing workcell. They would also prove to be invaluable tools when the "factories of the future" automate this entire product life cycle.

CHAPTER TWO

MANUFACTURING WORKCELL MODELLING AND TASKING FACILITY

Many products in the market are designed to improve the productivity of the manufacturing engineer. Tools such as computer aided design (CAD) facilities and task simulation tools are widely available and applied. But, there is a shortage of tools that help the manufacturing engineer take a product through its life cycle of product design, simulation and production in plant floor workcells. A few drawbacks exist due to the lack of an integration tool which ties together the above functions. The workcell models in simulation systems are not guaranteed to be accurate. The lack of data integrity in the models arise due to the models being created manually and the nonexistence of model verification accesses between the simulation systems and the actual workcell models. As long as the tools in the three phases of the manufacturing life cycle are tied together manually, chances for error cannot be eliminated. Errors occur due to improper translations of data between the tools for the various phases and the manual transport of data between them. Due to the lack of system integration, the simulation results become less desirable and less effective. Since the trust in the workcell data models is not complete and direct access of the workcells by simulation systems is not supported, the simulation results lose their predictive value.

Manufacturing workcells of today contain many classes of intelligent devices. Their device controllers are manipulated using the device native control languages. There are about as many control languages available as there are device manufacturers. This makes the job of a manufacturing engineer using multiple types of devices very difficult. Also, many vendors sell graphics simulation software packages which are geared towards manufacturing workcells [Mogal 1986, Hornick and Ravani 1986]. These packages perform simulations on workcell models which have been defined to them by the users. Generally, a restricted number of workcell models are defined for the simulation facilities. These models are painfully and slowly generated manually and are not guaranteed to be error free. Even though, over a period of time, the manufacturing workcell configurations could be changing, mechanisms are not available in the current systems for maintaining the integrity of the manufacturing workcell models with the actual workcells. The simulation systems usually have their own simulation languages. The simulation languages are very different from the native languages of the manufacturing workcell devices. Usually, no physical interconnection exists between the simulation system and the manufacturing workcell. So, even tasks which have been successfully simulated cannot be directly downloaded into the manufacturing workcells. They must be translated to the native languages of the workcell devices and loaded into them using other manual techniques. There are no fool-proof methods available for performing these translations and downloading into the workcells. The errors introduced in the translation and loading process may render the simulation results meaningless, although they do help in narrowing down the possibility of errors.

There exists a need for the manufacturing engineer to have one facility where the manufacturing devices can be defined and interconnected into a workcell and the tasks executing on these workcells defined and tested. The facility should permit the testing of the task both off-line and in the manufacturing workcell. The facility should be general enough to support the modelling and use of multiple device classes and types from the various manufacturers. The facility should also be language independent and easily extensible. It should allow device drivers to be installed for translating task plans into the device native languages and transferring code from the facility into the device controllers. The facility would allow tasks to be planned and tested in its task simulation interface. Once a task is "simulation successful" the task plan would be translated into the native languages of the workcell devices and downloaded and tested in the actual manufacturing workcells.

The above functions of a Modelling and Tasking Facility require large volumes of manufacturing device data for the workcell models and task specifications. Database researchers have been studying maintenance schemas for manufacturing databases [Su 1986, Kemper and Wallrath 1987]. From the perspective of the manufacturing engineer, there exists a need for

1) A language for describing manufacturing workcells to the data maintenance software,

2) Software which will permit error-free generation of the detailed derived knowledge about manufacturing devices from a higher level specification of the devices [Govindaraj and Doty 1986],

3) Simulation facilities for simulating tasks on the workcell models, and

4) Non-procedural and language independent task specification methods which are preferably workcell independent as well.

The Machine Intelligence Laboratory in the Department of Electrical Engineering of the University of Florida has been designing and developing a Manufacturing Workcell Modelling and Tasking Facility for the past four years. The overall goals and interactions of the facility are shown in Figure 2.1. This facility allows classes of devices to be described using a modelling basis and methodology developed for manufacturing workcells [Govindaraj 1987b]. The device models are used to drive knowledge generators which generate derived knowledge for the instances of pre-defined device models [Govindaraj and Doty 1986]. Device language independent tasking primitives [Govindaraj 1987a] and task planning strategies [Pal and Doty 1987] are also being studied under the Tasking portion of the facility. The Machine Intelligence Laboratory is collaboration with the Database Center of the University of Florida in studying the application of OSAM* for maintaining the manufacturing data generated and applied by the Manufacturing Workcell Modelling and Tasking Facility. OSAM* is an object-oriented Semantic Association Model for modelling data. This paper describes the design and current state of implementation of the Manufacturing Workcell Modelling and Tasking Facility.

2.1 An Example Scenario

We will look at a cooperating robot task in a multi-robot workcell, as a simple demonstration of the capability of the Workcell Modelling and Tasking Facility. Perceived data from a variety of sensors in the workcell could conceivably be used in achieving the required cooperation. For example, let us consider a simple workcell with two cooperating robots sharing a vision system. The shared vision system provides the synchronization data for cooperation between the two robots. The specified task is to transfer a workpiece from the workcell of manipulator1 to that of manipulator2. The vision system provides the detection of the arrival of the workpiece at the rendezvous pose where the exchange of the workpiece between the manipulators takes place. This simple static exchange cooperating task involves the following seven steps

1) Motion planning for manipulator1 to transfer the workpiece from its initial configuration to the rendezvous pose.

2) Sensor planning for the vision system to monitor the workcell to determine whether the workpiece has arrived at the rendezvous pose.

3) When the workpiece arrives at the rendezvous pose, the vision system should signal manipulator1 to stop and signal manipulator2 that the workpiece is available at the rendezvous pose.

4) Motion planning for manipulator2 to travel from its current configuration to the rendezvous pose.

5) Manipulator2 should grasp the workpiece at the rendezvous pose and signal manipulator1 that the workpiece has been acquired.

6) Motion planning for manipulator1 to release the workpiece and gracefully retract from the rendezvous pose.

7) Motion planning for manipulator2 to move from the rendezvous pose and continue with the rest of its programmed task.

The above simple task illustrates the application of the various modules of the Manufacturing Workcell Modelling and Tasking Facility for intra-workcell task planning and execution. We will later expand the example to illustrate inter-workcell task planning and execution. The above static exchange task involves three individually programmable devices (two manipulators and a vision system) in the workcell. The programs for the three devices and the synchronizations between the three programs have to be generated and executed. Although it may be relatively simple to generate the three individual programs, the synchronizations between them are not quite so simple. As the task gets more complicated, the planning of the independent device tasks and their synchronizations really gets out of hand. Also, with the variety of devices and control languages currently available for components in manufacturing workcells, even the generation of component tasks requires much user expertise. The above reasons indicate a need for a general purpose <u>Task Planner</u> for planning tasks for cooperating multi-robot workcells [Govindaraj 1987a, Pal and Doty 1987]. A five level hierarchy for describing tasks is outlined in Pal and Doty [1987]. The five levels are 1) Production Control Level, 2) Manufacturing Process Level, 3) Workcell Level, 4) Device

Level, and 5) Servo Level. The following properties are desirable in a task planner

1) The task planner should be capable of planning tasks for cooperating multi-robot workcells.

2) The tasks should be specified at levels higher than the workcell level.

3) The planner should be device independent in the task specification. For example, no device or control language dependencies can be introduced as part of the task specification.

4) The planner will recursively analyze the specified task into lower levels until reaching the workcell level.

The above properties stated as desirable for a task planner require the device dependencies to be driven to a lower interface level. This device interface is achieved through <u>tasking primitives</u>. The tasking primitives are device independent and provide the necessary building blocks for constructing multi-robot cooperation tasks. For general task planning in cooperating multi-robot workcells, we have identified three types of tasking primitives 1) Motion tasking primitives, 2) Sensor tasking primitives and 3) Control tasking primitives [Govindaraj 1987a]. In multi-robot workcells, the motion tasking primitives should handle single robot as well as multi-robot cooperating tasks. In

[Govindaraj 1987a] the motion tasking primitives for single as well as cooperating multiple robots are developed. These primitives are at the interface level and their execution can be realized by translating them into the corresponding device control languages. Primitives should be device independent and the device specifics built into the interface routines written for the realization of the primitives. The task planner should recursively analyze the high level task specifications into lower levels until the task trees consist only of the task primitives at the leaf nodes. The execution of the task primitives in the analyzed sequence gives the execution of the requested high-level task.

Tasks are specific to the workcell configuration for which they are planned. This workcell dependency of tasks requires that the planner have current models for all the workcells for which it is expected to plan, simulate and execute tasks. The Workcell Modelling Interface of the facility provides formal mechanisms for specifying classes and instances of devices in the workcells. Using this Interface, the user will be free to define object classes, such as the class of robot manipulators, and instances of this class, such as a puma-260. One of the problems with current off-line simulation facilities is that the data integrity of their workcell models are not assured. In our Workcell Modelling and Monitoring Facility, mechanisms will be provided for periodically calibrating the actual workcells and updating their models stored in the database. This will ensure the consistency of the workcell models with the actual manufacturing workcells. Simulating tasks using the monitored workcell models will give consistent results about the executability of the tasks in the actual manufacturing workcells.

The discussion so far has centered around manufacturing workcells and their models. The workcells have not been defined until now. In the next section, the general configurations of the manufacturing workcells will be defined and along with it the role of the cell controller in the manufacturing workcell.

2.2 Manufacturing Workcell

The Machine Intelligence Laboratory has been investigating workcells with multiple-cooperating robots [Doty 1983, Govindaraj 1987a]. The workcells considered in that research effort not only includes multiple robots, but also auxiliary support devices such as vision systems, tactile sensing systems, etc. The devices participate with one or all the robot manipulators during execution of the cooperating task. The workcells modelled by our facility, therefore, should include "objects" such as multiple robots, sensors and control devices.

Each manufacturing workcell in our model can have task cooperation among all the components of the workcell. We call this the intra-workcell cooperation. To achieve cooperation between different components in the workcell, we make use of a <u>cell controller</u> as a task sequencer. The workcells can also cooperate with other workcells at a floor or plant level of tasking interface [Pal and Doty 1987]. This type of cooperation is called inter-workcell cooperation. Inter-workcell cooperation is achieved by proper task planning and synchronization at planning levels above the workcell level.

This section describes the configuration of the typical manufacturing workcell modelled and the cell controller and its functions and responsibilities.

Manufacturing workcells contain sensed data distributed throughout the workcells and the factory floor. This sensed data can be manipulated and interpreted at different levels of abstraction, depending on the level of information content in the data. Sensed data in their different abstraction levels are used by the various task planning, execution and monitoring modules in the workcells. The different levels of abstraction of the sensed data, collectively, are called the <u>Perceived World Model</u>. Two important issues are whether to centralize or distribute the perceived world model, and what role should the cell controller take in maintaining the perceived world model in a manufacturing workcell.

Figure 2.2 shows a typical manufacturing workcell and the interactions the cell controller has with the two main software modules of the Manufacturing Workcell Modelling and

Tasking Facility to be discussed in a later section. Devices 1 through N in the figure can be any component that is a part of the manufacturing workcell.

Cell Controller

The cell controller in the manufacturing workcell is the central controlling resource in each workcell. The cell controller responsibilities include

1) Routing task requests to various devices in the workcell,

2) Synchronizing task executions between the various workcell components,

3) Monitoring task execution, both at the device and cell levels,

4) Maintenance of the perceived world model,

5) Communication with other cell, plant floor and facility level controllers, and

6) Calibrating and monitoring workcells and providing the workcell monitoring interface with updated workcell models.

We do not go into the design of specific cell controllers in our research. Our interest is to define logically the rationale behind the functions of the cell controller and the criteria that the cell controllers in our model of the manufacturing workcell facility should satisfy.

As our example scenario illustrates, even for simple multi-robot cooperation tasks, there are many elements of individual motion, sensor and control planning for the workcell components involved. For successful cooperation, the individual tasks of the various components in the workcells should be properly synchronized. The Task Planner is independent of the task execution in the workcells. Therefore, the Task Planner can be shared between the various workcells in the facility. The planner can analyze and plan tasks successfully as long as the workcell models for the manufacturing facilities used in the task are available for the planner. However, during task execution, the progress of the various tasks in the workcell devices and the synchronization between them have to be monitored more carefully. This necessitates the incorporation of a separate cell controller into each of the manufacturing workcells.

Perceived World Model

The world model in the database comprises of two kinds of data. One is the user generated world model which is created by the user or generated from the data provided by the user. Usually, the data for the user generated world model does not involve any sensory devices. The second type is the perceived world model. The perceived world model is generated completely from the sensory data from the devices in the manufacturing workcell. The sensors in the manufacturing workcells provide the capability of monitoring the manufacturing workcells. Sensors act as perception devices for perceiving the state of the manufacturing workcells.

At any instant of time, the perceived world model contains the current sensed state of the workcell. Various parts of the perceived world model data can come in at different update rates. The update rates depend upon the sensory devices from which the data originate. For example, the perceived world model includes data from position and velocity sensors of robot joints which are updated rapidly to the sensed and computed data of several special purpose sensors such as the vision systems, touch sensors, etc. which usually changes at a much slower rate.

The function and application of the sensory data depends on its level of abstraction. We identify sensed or perceived world data to be at four hierarchical levels of abstraction. From the lowest to the highest, the levels in the perceived world data are 1) Raw Sensory Data, 2) Feature Data, 3) Recognition Data, and 4) Relationship data. Each higher level of sensory data can be generated from a lower level using computations to process the lower level sensory data. These computations can be embedded directly in the sensory devices themselves. Therefore, sensory devices can provide data for the perceived world model in any subset of its four identified layers.

To give examples for the four layers of the perceived world model, we will choose data from a vision system. The vision camera provides <u>raw sensory data</u> at binary pixel level. The binary images can be processed to extract the edges and other boundary and feature data from the binary images. These data constitute <u>feature data</u> of the vision system. Feature data from the vision system can be used by template matching and other algorithms to match the perceived data with stored template data. These algorithms yield vision <u>recognition data</u> to identify perceived objects. The perceived and recognized objects can then be placed into classifications and categories which associate the perceived objects with other objects, thus creating vision <u>relationship data</u>.

The perceived world model is a result of sensed data and the processing of sensed data. In designing the Workcell Modelling and Tasking Facility, the issue of whether to place this perceived world model locally within each workcell or globally in the plant floor or facility level controllers is critical. We decided to distribute and partition the perceived world model according to the generation and application domain of the sensed data. Most often, the perceived world model is only applicable locally within the workcell. Also, the centralization of the perceived world data would tremendously increase the data flow in the facility between the various workcell controllers and the central perceived world model manager. Due to these two reasons, we decided to keep all the locally generated and applied sensory data in their respective workcells and distributed throughout the facility. The globally applied perceived data would be sent to a central store at slow update rates or as queried updates. The task of processing the local perceived world model data and maintaining it is entrusted to the cell controllers in the manufacturing workcells.

2.3 Workcell Modelling And Monitoring Interface

The Manufacturing Workcell Modelling and Tasking Facility comprises of two major modules (Figure 2.3). The first module is the Workcell Modelling and Monitoring Interface and the second is the Workcell Tasking Interface. In this section, we discuss the Workcell Modelling and Monitoring Interface.

The primary functions of the Workcell Modelling and Monitoring Interface are to create and maintain the data integrity of workcell models. The following are the more detailed responsibilities of the Workcell Modelling and Monitoring Interface

1) Define the object models, basis and aggregates to describe classes of components in the manufacturing workcell [Govindaraj 1987b]. A

manufacturing workcell modelling language will be used as an expert manufacturing professional's interface to describe the above models. For example, classes of components such as robots, sensors, controllers, etc. can be defined by the manufacturing expert using this language and the modelling interface. The modelling language will permit the definition of the properties of the device classes, operations permitted on the classes, rules for generating the derived knowledge about the members of the classes and the restrictions on the classes of objects.

2) Instantiate the data for the members of pre-defined classes of components. In creating data for instances of the class robots, the users could specify and create data for defining, for example, a Unimation Puma-260, or a Cincinnati-Milacron T3 robot manipulator. The user interface for the collection of specification data and the generation of derived data for the class instances are driven by the pre-defined models for the classes to which the instances belong [Govindaraj and Doty 1986].

3) Maintain the integrity of the data models in the manufacturing database. This is the function of the Workcell Monitoring Interface and is

applicable only when an actual workcell exists corresponding to the workcell model created in the manufacturing database. The facility also permits workcell models to be created to study layouts and tasks for nonexistent workcells. The Workcell Monitoring Interface is not applicable to such "mythical workcells." The Workcell Monitoring Interface invokes calibration routines in the actual manufacturing workcell to check and update the workcell model in the database. These periodic calibrations help to maintain conformity between the actual manufacturing workcell and their database models.

Based on their functions, the Workcell Modelling and Monitoring Interface can be divided into the Workcell Modelling Interface and the Workcell Monitoring Interface. The Workcell Modelling Interface (Figure 2.4) is primarily a user interface module to describe manufacturing component classes, instances of these classes and their interconnections to form manufacturing workcells. The Workcell Monitoring Interface (Figure 2.5) mainly interacts with manufacturing workcells to calibrate them and update their models stored in the manufacturing databases.

Workcell Modelling Interface

The primary objective of the Workcell Modelling Interface is to provide a human interface for defining the classes of components possible in the manufacturing workcells, instances of pre-defined classes and the interconnection of these instances to form workcells, plant floors, facilities, etc. The Workcell Modelling Interface makes a distinction between its possible users. There are the Expert Users of the interface and the normal Users. The Expert Users in this case are experts in manufacturing. All users can generate instances for the objects in the manufacturing database. Only Expert Users are allowed to define the object models, basis and aggregates in the manufacturing databases. The Expert Users are provided a Manufacturing Workcell Description Language [Govindaraj 1987b] which can be used to define the object classes. This language recognizes five basis objects in the manufacturing workcell: statica, sensors, actuators, informata and processes. This is a functional basis and not a primitive set of objects. The functional basis can define the properties of the objects, operations permitted on the logical representation of the objects and the rules and processes used to derive further derived knowledge about the objects. The language also permits aggregations of the basis set. The aggregates permit other object descriptions to be built based on the functional basis and/or other pre-defined aggregates. Aggregations are controlled by aggregation rules and aggregation restrictions. The modelling language provides some of the aggregations useful in manufacturing workcell modelling and also gives users the capability to define their own aggregations. Using a programming language analogy, the basis is like the data types defined in the language and aggregations are mechanisms provided for user defined data types based either on the basis as the data type or other pre-defined aggregations as the user defined data types of the language.

The Expert User will use the Manufacturing Workcell Modelling Language to define new classes of manufacturing workcell objects using aggregations. Expert and normal Users may define data for the instances of pre-defined basis, aggregates or newly defined aggregates in the facility. Instances cannot be generated for classes of objects that are not pre-defined by the Expert User. Data collection and derived data generation for the modelling instances are controlled by the object models in the Workcell Modelling Interface. The Workcell Modelling Interface permits the modelling and instantiation of components in workcells, workcells and other higher aggregations based on components and the workcells. The Modelling Interface also permits the instantiation of Parts into the Parts database. Parts can be modelled using the basis object statica. Parts are placed in a separate database since they are the workpieces manipulated to produce useful work in the manufacturing workcells and they are not a part of the manufacturing workcell.

Workcell Monitoring Interface

One of the drawbacks of currently available off-line manufacturing workcell simulation facilities is that they do not provide the capability to maintain the data integrity of the workcell model. Changes in the actual manufacturing workcells are not carried out on their data models in the simulation systems. The user of the task simulator, therefore, is not assured of task feasibility even if simulations so indicate.

The Workcell Monitoring Interface addresses this problem by overseeing the workcells and maintaining the integrity of their data models when workcell changes are detected. Workcell calibration routines built into the cell controllers have the responsibility of maintaining an accurate view of the workcell that they control. Calibration requests sent by the Workcell Monitoring Interface to the cell controllers demand updates of the data models in the workcell and parts database. The preservation of data integrity of the manufacturing workcell models gives the users more confidence in the simulation results. Simulations now give a more accurate and current view of whether the tasks planned and submitted to the workcells will work in the manufacturing workcells or not.

2.4 Workcell Tasking Interface

The Workcell Tasking Interface is the second major module of the Manufacturing Workcell Modelling and Tasking Facility. The Tasking Facility is responsible for planning, testing, executing and monitoring tasks being executed in the manufacturing workcells. The tasking interface processes task requests from users or from a database such as OSAM*.

OSAM* rule processing facilities have the capability of originating task requests that change the state of the workcell [Su and Raschid 1985]. This change of state of the workcell also results in changes in the world model of the workcells. Without the Workcell Tasking Interface, once OSAM* issues a rule-based task request, it cannot monitor the proper processing of the task request. It is usually assumed that the task requests are issued to perfect manufacturing workcell which can process all requests issued by OSAM* and that the issued requests will be properly executed. The tasking interface will provide capabilities for testing the feasibility of task requests and the monitoring of the tasks until completion.

Processing of task requests in the Workcell Tasking Interface is a four step process 1) Task Planning, 2) Task Simulation, 3) Task Execution, and 4) Task Confirmation. This four step task processing is driven by the types of workcells and task executions being studied by the Machine Intelligence Laboratory (MIL). MIL is currently studying cooperative task executions in multi-robot workcells. These tasks require individual task planning for the devices participating in the task and proper synchronizations in their task executions. This is not easily achieved--particularly for complex workcell configurations and task requests. Therefore, we are studying automated modelling and task planning methods [Govindaraj and Doty 1986, Pal and Doty 1987, Govindaraj 1987a, Govindaraj 1987b]. Even in single robot workcells with a high possibility of collisions, simulations have proved to be a valuable tool for establishing off-line the feasibility of the task request. The value of simulations is increased in multi-robot workcells where the workspace environment is dynamic, increasing many-fold the possibility of collisions. In our Tasking Facility, we will use the Task Simulator to verify the task plans generated by the Task Planner. Obviously, if the task does not succeed in the simulator, it is definitely not going to succeed in the manufacturing workcell. The task requests will be passed through multiple iterations between the planner and the simulator until a "simulation successful" task plan is generated. The simulation successful task plan will be submitted to the Task Executor for execution in the manufacturing workcell. The Task Confirmer monitors the task execution and sends the task confirmation or failure code to the initiator of the task request.

The Workcell Tasking Interface and the associated data modules are shown in Figure 2.6. The Tasking Interface processes task requests issued at various levels. Pal and Doty [1987] identify five levels in the task planning hierarchy 1) Production Control Level, 2) Manufacturing Process Level, 3) Workcell Level, 4) Device Level, and 5) Servo Level. The simulations for studying task feasibility work at the workcell level. Commands issued to the cell controllers for routing and synchronizing are at this workcell level. The simulator also supports simulation of individual devices and the lower servo level control of the devices. As far as task simulations are concerned, workcell level is sufficient. Device and servo levels are used to build the simulation library and permits the Expert User to test devices and device control algorithms.

For the task planner to generate plans for high level tasks, the workcell models generated earlier by the Workcell Modelling and Monitoring Interface are essential. The task plans are successively analyzed task specifications until the leaf nodes of the task trees reach one of the tasking primitives [Govindaraj 1987a]. The tasking primitives for the motion planning of single and multi-robot workcells are currently being studied by the MIL research group.

For tasks above the workcell level, higher level heuristics are required for task partitioning and task assignment to workcells. These heuristics will be available to the task planner through data in the Manufacturing

Procedures and Task Decomposition Rules portions of the manufacturing database.

We continue in this section with a more detailed examination of the various modules in the Workcell Tasking Interface.

Task Planner

The Task Planner (Figure 2.7) is at the heart of the Workcell Tasking Interface. The task planner can accept requests from the User directly or from the rule-processing portion of OSAM*. The Task Planner recursively analyzes the task specification until reaching the workcell level. The task is further analyzed at the workcell level until the leaf nodes of the generated task tree contain only the tasking primitives [Govindaraj 1987a]. The tasking primitives are independent of the device specifics and can be used to execute tasks on general workcells of known configurations. The tasking primitives act as an interface providing the facility with the device independence required for general purpose task planning. They also provide devices with their command language specifics required for executing tasks on specific devices. Since cell controllers will be issued commands and synchronization requests at the workcell level, we decided to keep simulations also at the same level. The cell controller is responsible for routing the individual task requests to the appropriate devices and handle the synchronization between the task requests of the individual devices.

In planning tasks, workcell models provide the planner with the current workcell configurations. The task planner also uses the manufacturing procedures and the task decomposition rules. The manufacturing procedures are high level descriptions of the capabilities of manufacturing workcells and devices. Task decomposition rules lay down the heuristics for partitioning tasks from the manufacturing process level and assigning them to the workcells. The planner generates lower level task trees from the input task specifications. The task trees generated by the planner can be used by the simulator for off-line verification of the tasks before the task plans are submitted to the task executors. The user or OSAM* should also have the option of requesting the task planner to analyze the task only into the next lower level instead of the workcell level. This option allows various aspects of the task planning process to be studied. This includes the functioning of the task planner, the validity of the task planning rules, etc.

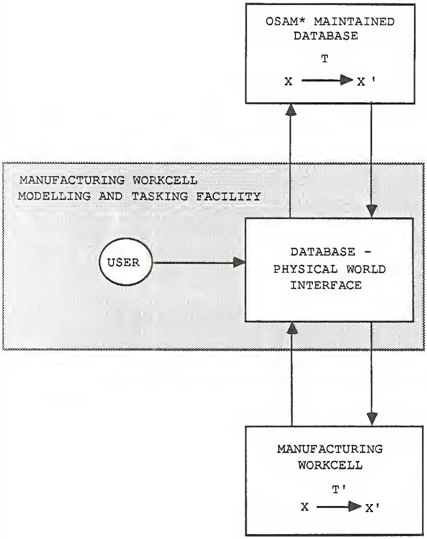
Task Simulator

The Task Simulator is used to verify task plans generated by the Task Planner in the Workcell Tasking Facility. For the most part, the simulator is assumed to function at the workcell level. The only exceptions are for the expert user who is allowed to perform device and device servo simulations at lower levels. The Task Planner generates task trees for task requests submitted to them by the users or by the rule-based task generation portion of OSAM*.

The Task Planner and Task Simulator work in a tight loop as shown in Figure 2.8. The decomposed task trees from the planner are submitted to the simulator. The simulator performs a task simulation of the task trees on the workcell models of the manufacturing workcells for which they were generated. The task simulator requires access to the tasking primitives for two reasons. Firstly, the task trees constructed by the Task Planner have tasking primitives at the leaf nodes. The Task Simulator requires these primitives for performing task simulations. The second reason is that the simulator is designed to work at the primitives level to keep the simulator independent of the specific device dependencies. The one and multi-robot workcell tasking primitives will be accessible to the task simulator through the tasking primitives database. The Task Planner will use the Task Simulator results in constantly modifying its task plan until it arrives at a "simulation successful" task plan. This successfully verified plan will be used in the manufacturing workcell for task execution.

Task Executor and Task Confirmer

The Task Executor and Task Confirmer are used to execute and monitor task executions in the manufacturing workcell. The Task Executor and Confirmer reside in the manufacturing workcell controller. The Task Executor will be given the "simulation successful" task plan or task tree for execution. The executor in the cell controller has the responsibility of distributing the planned task to the individual devices in the multi-robot manufacturing workcell and synchronizing the task execution of the various devices in the workcell. While the tasks execute, the Task Confirmer monitors the execution to detect any failures. If there are failures in the task execution, the watchdog monitor stops execution of the task, shuts down the workcell and informs the task initiator and the Task Planner about the failure of the task. The Task Planner then has to take alternative action to plan a new task tree for the specified task after initiating actions to cleanup the workcell from the previous failure and get the workcell into an acceptable state to continue task execution. Whether the task succeeds or fails, the task initiator is sent the success or failure code indicating the status of the task. If the task requests are initiated by OSAM* rules, the task execution rules may have error recovery rules built-in. These error recovery rules will initiate new tasks which will also have to be planned through the Task Planner and go through the previously described tasking iterations. If the Task Confirmer sends a success code to OSAM*, then the success code may drive functions in the database to update the data models to reflect the current state of the world model. This way, OSAM* will be given the capability of initiating task requests, planning and monitoring task executions and updating the database only when the task succeeded in the manufacturing workcell without assuming the success of the task request.


OSAM* MAINTAINED DATABASE
T
X → X'
MANUFACTURING WORKCELL MODELLING AND TASKING FACILITY
USER
DATABASE - PHYSICAL WORLD INTERFACE
MANUFACTURING WORKCELL
T'
X → X'

* Maintain accurate view of Physical World
* Change view when Physical World changes
* Drive changes in Physical World (changes in data should result in changes in Physical World)

* Provides the interface between the DB and the Physical World and vice versa
* Functions include
  - describe physical world to DB
  - monitor physical world and update changes to DB model
  - define, execute, control and monitor task execution in the physical world

* Target application
* Produce useful work

FIGURE 2.1 Manufacturing Workcell Modelling and Tasking Facility

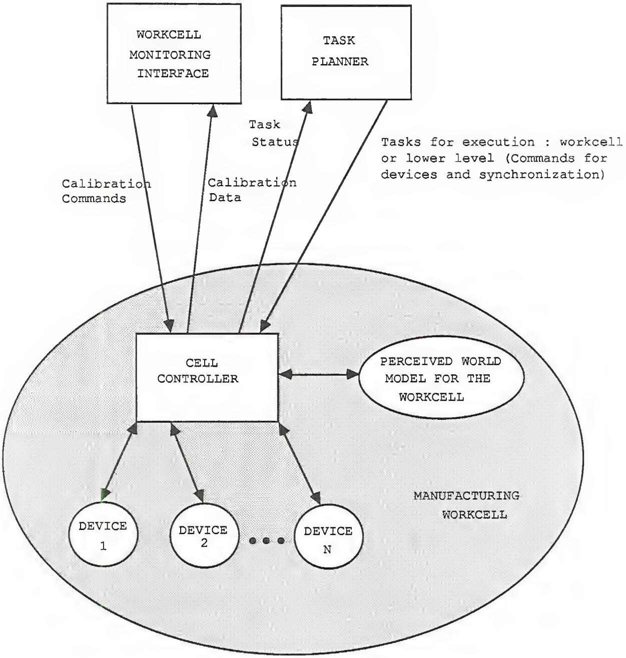

WORKCELL MONITORING INTERFACE
TASK PLANNER
Task Status
Tasks for execution : workcell or lower level (Commands for devices and synchronization)
Calibration Commands
Calibration Data
CELL CONTROLLER
PERCEIVED WORLD MODEL FOR THE WORKCELL
MANUFACTURING WORKCELL
DEVICE 1
DEVICE 2
DEVICE N

FIGURE 2.2 Manufacturing Workcell

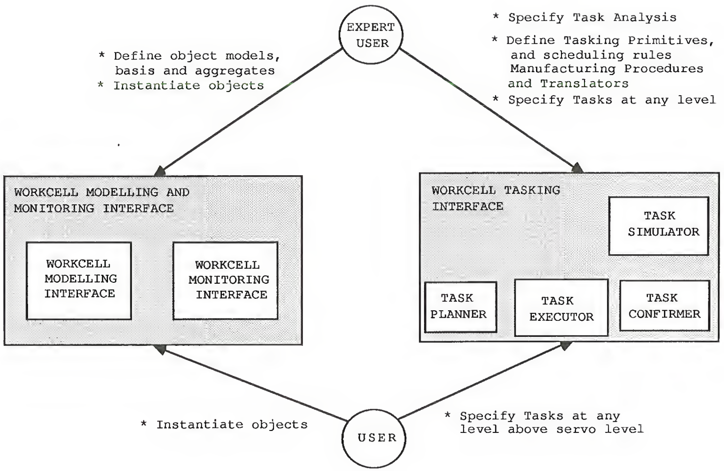

EXPERT USER
* Define object models, basis and aggregates
* Instantiate objects
* Specify Task Analysis
* Define Tasking Primitives, and scheduling rules Manufacturing Procedures and Translators
* Specify Tasks at any level
WORKCELL MODELLING AND MONITORING INTERFACE
WORKCELL MODELLING INTERFACE
WORKCELL MONITORING INTERFACE
WORKCELL TASKING INTERFACE
TASK SIMULATOR
TASK PLANNER
TASK EXECUTOR
TASK CONFIRMER
* Instantiate objects
USER
* Specify Tasks at any level above servo level

FIGURE 2.3 Main Modules of the Manufacturing Workcell Modelling and Tasking Facility

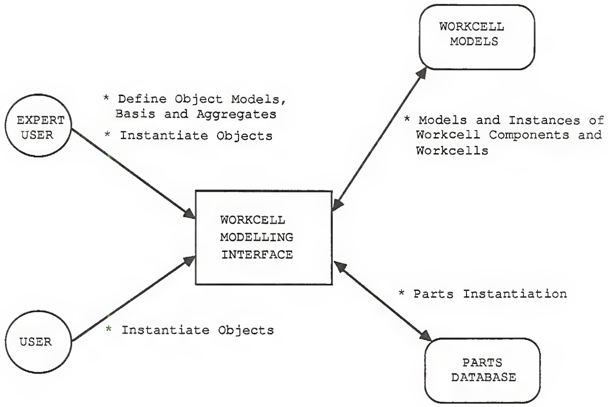

WORKCELL MODELS
EXPERT USER
* Define Object Models, Basis and Aggregates
* Instantiate Objects
* Models and Instances of Workcell Components and Workcells
WORKCELL MODELLING INTERFACE
* Parts Instantiation
USER
* Instantiate Objects
PARTS DATABASE

FIGURE 2.4 Workcell Modelling Interface

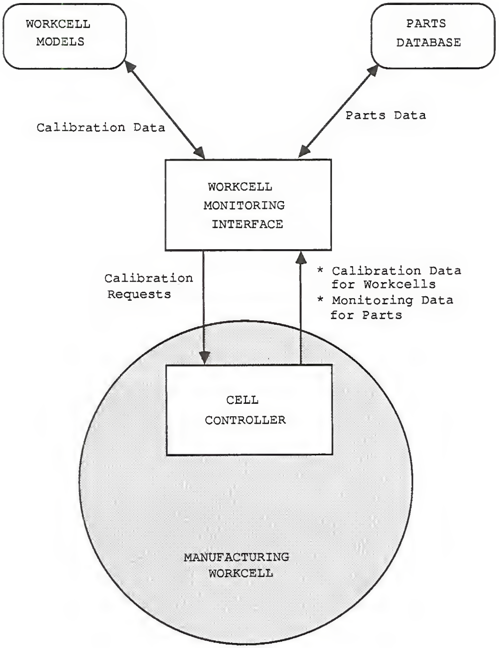


FIGURE 2.5 Workcell Monitoring Interface

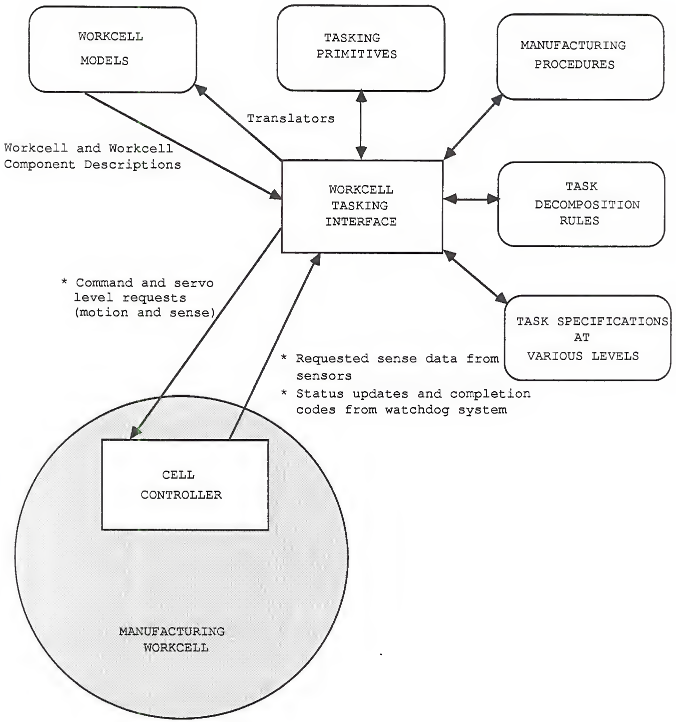

WORKCELL MODELS
TASKING PRIMITIVES
MANUFACTURING PROCEDURES
Translators
Workcell and Workcell Component Descriptions
WORKCELL TASKING INTERFACE
TASK DECOMPOSITION RULES
* Command and servo level requests (motion and sense)
TASK SPECIFICATIONS AT VARIOUS LEVELS
* Requested sense data from sensors
* Status updates and completion codes from watchdog system
CELL CONTROLLER
MANUFACTURING WORKCELL

FIGURE 2.6 Workcell Tasking Interface

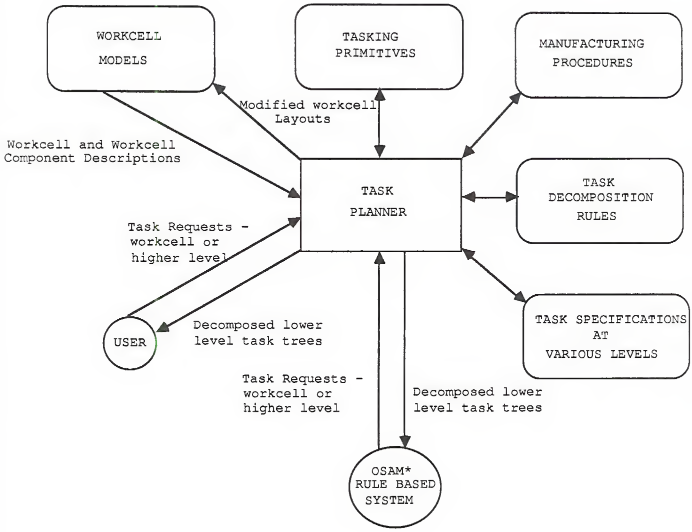

WORKCELL MODELS
TASKING PRIMITIVES
MANUFACTURING PROCEDURES
Modified workcell Layouts
Workcell and Workcell Component Descriptions
TASK PLANNER
TASK DECOMPOSITION RULES
Task Requests - workcell or higher level
USER
Decomposed lower level task trees
TASK SPECIFICATIONS AT VARIOUS LEVELS
Task Requests - workcell or higher level
Decomposed lower level task trees
OSAM* RULE BASED SYSTEM

FIGURE 2.7 Task Planner

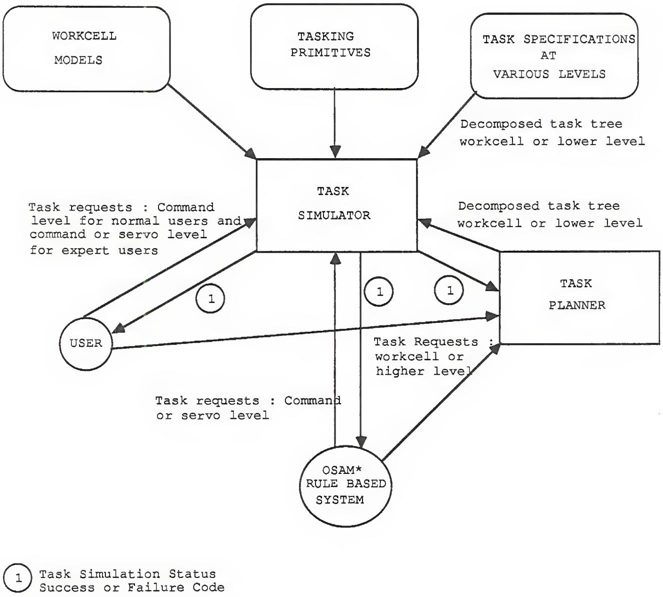

WORKCELL MODELS
TASKING PRIMITIVES
TASK SPECIFICATIONS AT VARIOUS LEVELS
Decomposed task tree workcell or lower level
TASK SIMULATOR
Task requests : Command level for normal users and command or servo level for expert users
Decomposed task tree workcell or lower level
TASK PLANNER
1
1
1
USER
Task Requests : workcell or higher level
Task requests : Command or servo level
OSAM* RULE BASED SYSTEM
1 Task Simulation Status Success or Failure Code

FIGURE 2.8 Task Simulator

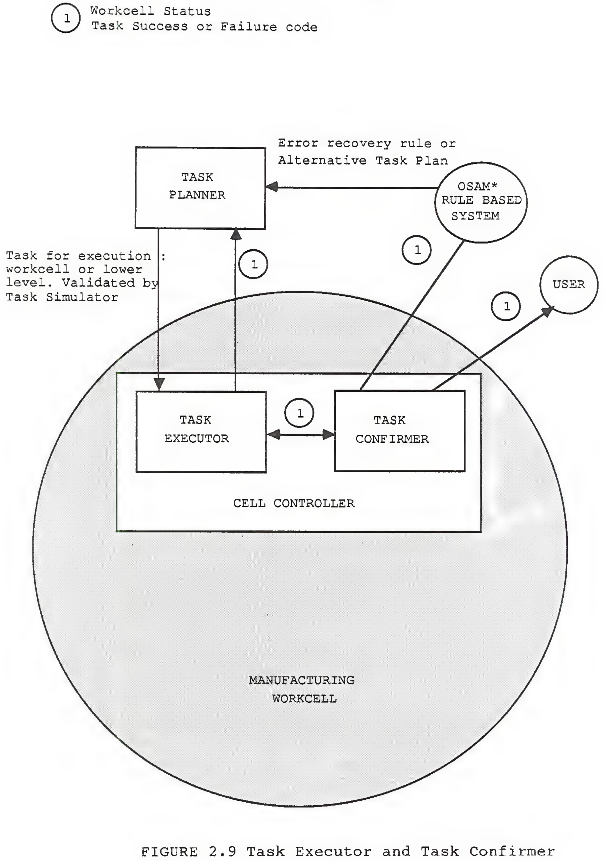


FIGURE 2.9 Task Executor and Task Confirmer

CHAPTER THREE

A MODELLING BASIS FOR MANIPULATOR WORKCELLS

In Chapter Two, we discussed the product development life-cycle of manufactured products. We also looked at the various software products available to a manufacturing engineer and the need for system integration of these tools. An engineer involved in product development has to learn all these tools and their languages. Therefore, a manufacturing engineer also has to have a high level of expertise in computer systems.

In this chapter, a language aimed specifically towards manufacturing engineers is developed. Using this language, users can specify physical and logical workcell layouts, and specify, test and execute tasks in the described workcells. The language is based on a set of five classes of objects found in manufacturing workcells. This language can be used as a manufacturing expert interface, regardless of the underlying data management and application scheme. We call this language the Manufacturing Engineers' Description Language (MEDL). Figure 3.1 shows the intent of MEDL.

A manufacturing engineer uses MEDL to describe a high-level view of a physical manufacturing facility. A MEDL Interpreter (MEDLI) converts this MEDL description of the manufacturing facility into the representational scheme of the underlying database systems. This allows manufacturing engineers to "port" workcell models between different facilities, regardless of the underlying application scheme. The application programs only have to support MEDLI to permit porting the modelling data. Consequently, a manufacturing engineer only need learn MEDL to design and test manufacturing facilities.

At the MIL, current investigations employ OSAM* as an underlying database schema for the Manufacturing Workcell Modelling and Tasking Facility. An implementation of MEDLI, in this case, will convert from MEDL to the s-diagram representation of the models in OSAM* [Su et al. 1988] and, thence, to OSAM* "objects."

Figure 3.2 shows an application scheme for MEDL and MEDLI. Chapter Five describes a Knowledge Generator for devices in manufacturing facilities. The knowledge generators derive detailed knowledge about devices based on their high-level specifications. MEDL can be used to provide the high level specifications of device classes. The object models are manufacturing experts' descriptions of classes of devices. The user specification entries and user instantiations generate specific instances for the device classes. For example, the expert user will use MEDL and MEDLI to set up the object model and derived knowledge generators for a class of devices called kinema. A user who wishes to model a kinema instance, for example the PUMA-260, will define the high-level details of the instance. The kinema knowledge generators can then be invoked to generate the detailed derived knowledge for the kinema instance. This derived knowledge will be used by application programs such as simulation and control programs.

MEDL permits objects to be modelled at any level of detail of the users' choosing. The aggregations can be set up in MEDL to restrict the level of detail at which the modelling is to be performed. For example, suppose a 6 dof robot manipulator with 6 links and 6 joints is being modelled along with its controller. Each of the revolute joints also has actuators which drive them. The controller has matching servo-control boards for each actuator of the manipulator. The controller receives host computer requests for achieving specified configurations of the manipulators and computes the set points for the different joints to achieve that configuration. These set-points drive servo controller boards which in turn drive manipulator actuators to achieve the commanded joint space configuration. A user modelling the interactions at the command and configuration level need not know the lower level detail of the control. In this case, the user can set up the manipulator model to be a kinematic chain which can achieve configurations sent to the controller. The manipulator, therefore, needs to be modelled only at the kinematic level and the controller at the command level. Users who desire lower levels of detail can build detailed aggregations to support their view.

In this chapter, we provide a basis on which the modelling primitives are built. We also give aggregation rules with which users can set up object models for workcell device classes. The basis and aggregations are illustrated with various examples. Appendix 1 gives a concise description of MEDL.

3.1 Notation

Before describing the modelling basis and aggregation rules that constitute MEDL, we describe the notations used in the language. The Backus Naur Form (BNF) serves as a basis for our notation. Wherever BNF is inadequate to describe MEDL, extensions to BNF are provided. The notational basis for describing MEDL is, therefore, an extended BNF. BNF is a conventionally accepted notation for describing computer languages. So, it is natural to use an extended BNF to describe MEDL.

Implicit in this notation is a left precedence rule in the definition of operators. Further, upper-case words will imply terminal tokens in the BNF descriptions and lower-case words will define non-terminal tokens. Terminal lists described in this dissertation represent some common data required in such lists. Additions and extensions to terminal lists are permitted and are expected to be system installation dependent. Below are some of the notational constructs used in MEDL.

BNF Rules

That M can be obtained from a or by a operating on b is indicated by

```
M ::=   a ¦ a <op> b
```

Operators are enclosed in angle-brackets to distinguish them from other terminal, or non-terminal tokens. MEDL operators will be described along with the objects they operate on and will be discussed where appropriate.

Aggregation Parameters define attributes for aggregate objects. This is indicated by square brackets as [aggr_param]. Optional parts of the descriptions are enclosed in parentheses: (optional descriptions). Comments in MEDL are enclosed in curly braces and can cross line boundaries. Example, {This is a comment}.

UNIX's Yet Another Compiler-Compiler (yacc) is a BNF based language parser generator. Yacc was used to test the syntactical structures of MEDL. The MEDLI implementation currently under investigation at the MIL will also use yacc.

3.2 Program Structure of MEDL

MEDL is intended primarily for building object models for manufacturing workcell device classes. It provides expert manufacturing users a vehicle for controlling data generation for device instances in the described classes. MEDL is envisioned primarily as an interactive language where the expert user sits at a display terminal and specifies the structure and data to be collected to define instances of classes of manufacturing devices. During instantiation these object models will drive the data collection and generation software to acquire the required data. It is also foreseen that the automation of the Task Planner and the workcell layout generator may present a requirement where MEDL descriptions may be submitted as a batch file. Even in an interactive use of MEDL, the assumption is that any object model used to declare instances during the declarative part should have been pre-described before use as an object type. In either case, the "program" structure of MEDL is

```
DESCRIPTIONS:
list_of_aggregations
DECLARATIONS:
list_of_instantiations
```

The above MEDL program structure shows that MEDL has a descriptive part and a declarative part. The descriptive part provides MEDL the power of building manufacturing workcell object models from an identified basis and other pre-defined object models. In the declarative part, the user can associate named object models to object descriptions made in the descriptive part of MEDL. MEDL allows the user to build as well as to instantiate object models. These two functions of MEDL are akin to declaring user defined data types and variables in high level programming languages. A manufacturing expert user building object models of robots, for example, will use the descriptive part of MEDL. An expert or a normal user using the robot object model to guide the description of robot instances, for example Puma-260 or GP-66, will use the declarative parts of MEDL. In this dissertation the focus is on the descriptive part of MEDL only and not the declarative portions. The syntax for the declarative part is defined, however, and examples are developed.

The descriptive and declarative parts of MEDL were defined using list_of_aggregations and list_of_instantiations which are detailed as shown below.

```
list_of_aggregations ::=   aggregation
                      | list_of_aggregations aggregation

list_of_instantiations ::=   instantiation
                      | list_of_instantiations instantiation
```

The non-terminals <u>aggregation</u> <u>and</u> <u>instantiation</u> in the program structure above will be elaborated as we proceed with the description of MEDL. In order to provide motivating examples, the declarative part of MEDL is presented before defining the descriptive part.

3.3 Declarative Part of MEDL

We begin this discussion with the syntax of the declarative part of MEDL and illustrate the concepts with some examples.

Declarative Syntax of MEDL

```
instantiation ::= instance_list ':' instance_type ';'

instance_list ::=   instance_name
                  ¦ instance_list ',' instance_name

instance_name ::= identifier

identifier ::=   letter
               ¦ identifier letter_or_digit

letter ::= lower_case_letter ¦ upper_case_letter

digit ::= '0' ¦ '1' ¦ ...... '9'

underscore ::= '_'

letter_or_digit ::= letter ¦ digit ¦ underscore

instance_type ::=   basis
                  ¦ aggregate_name

aggregate_name ::= identifier
```

Names which are legal strings of a letter followed by a letter, digit or underscore can be used as MEDL identifiers. The identifiers can be a list and will be of type defined by the instance_type. The legal types in MEDL are any of the basis types to be defined in the next section, or any user defined aggregations which are object model declarations. The object model declarations should be available for the language during the instance declarations. MEDLI should be built with mechanisms to keep track of all the legal aggregate names or object models during instantiation. Below are some examples for the declarative constructs of MEDL.

Examples of MEDL Declarations

```
Table_1, Table_2       : statica;
optomation_controller,
  p50_robot_controller : controller;
```

These examples demonstrate the usage of basis and aggregate names for instantiation. Table_1 and Table_2 are defined as type statica, a basis element to be described in the next section. Controller, on the other hand, is a user-defined aggregation and, in the above example, optomation_controller and p50_robot_controller are declared to be controller types.

3.4 Functional Basis

Manufacturing workcells contain two types of entities: 1) physical or concrete entities and, 2) logical or abstract entities. Concrete entities have a physical form and other physical properties attached to them. They occupy space in manufacturing workcells. All concrete objects must have a body attached frame upon which its geometric definition hinges. Spatial relationships between bodies are described by the coordinate transformations existing between these body attached frames. Abstract entities, on the other hand, do not occupy space or have physical properties. But, they may be present in or managed by concrete objects. Abstract properties are logical representations of information.

There are many ways to represent information. Of these, we choose to use the data structures of the implementation programming languages as the representational scheme. This choice is primarily due to the fact that computer tools and languages will be used to model and maintain the information. Instead of restricting the implementer to use a certain system or programming language, we choose to allow the MEDL implementer to use his or her choice of system and language to describe abstract entities in MEDL. The following statements define some properties of the concrete and abstract domains.

<u>Properties of Concrete and Abstract Domains</u>

```
concrete_properties ::= geometry   color     texture
                        material   WEIGHT    DENSITY

geometry ::= shape   VOLUME   AREA   INERTIAS
                     BODY_ATTACHED_FRAME

shape ::= data_structures defining shapes
                (implementation CAD system dependent)

color ::=  data_structures defining color
                (implementation CAD system dependent)

texture ::= data_structures defining texture
                (implementation CAD system dependent)

material ::= data_structures defining material
                (implementation CAD system dependent)

abstract_properties ::= data_structures
                (object-oriented data structures or
                 data structures based on the
                 implementation programming language)
```

While studying the functional properties of objects in the manufacturing workcells other basis objects are possible. This functional basis is shown in Figure 3.3. <u>Statica</u> are purely concrete objects and <u>Informata</u> are purely abstract objects. Workcells contain objects which are transducers between the concrete and abstract domains. The transduction can be achieved in both directions. *Sensors* are transducers from the concrete to the abstract domain and *actuators* are transducers from the abstract to the concrete domain. Sensors and actuators have both concrete and abstract properties. Therefore, for sensors and actuators, both the physical and logical properties have to be modelled.

The last of our functional basis is a *Process* which models informata transformational aspects of objects. Notationally, the basis can be defined as

MEDL Functional Basis

```
basis ::=   statica
          ¦ informata
          ¦ sensor
          ¦ actuator
          ¦ process
```

Every member of the basis has either a descriptive (noun) part, an action (verb) part, or both. In our modelling basis, statica and informata lie purely in the concrete and abstract domains respectively. Statica and informata have only a descriptive part. Due to their transduction nature, sensors and actuators have both, descriptive and action parts. Processes are in the abstract domain only. They provide the transformations that acts on informata, sensors and actuators. Since processes do not have any physical properties, they only have an action part.

This section presents the five functional basis mentioned above. Each member of the basis will be defined with examples and the operations possible on them will be identified.

Statica

Statica is used to model rigid-body objects. Statica are solid objects whose physical properties define them. Statica objects cannot have movable sub-structures. If an object has movable sub-structures, each sub-structure should be modelled using statica and the whole object modelled using aggregates (see section 3.5). Statica objects can have any complex shape or size. Physical attributes are the only defining parameters for statica. Functionally, the only "action" of statica in a manufacturing workcell is to consume space. Notationally, statica can be defined as shown below:

Statica Definition

```
statica ::= concrete_properties
```

Many of the geometric properties of statica can be derived from solids geometric modelling tools provided by most CAD systems. Other terminal tokens can be added to statica_attributes and geometry to enrich their representational scheme as desired by application programs.

All statica must have a body attached frame describing them. The other non-terminals, shape, color, texture and material, are dependent on the application systems. These properties are related to the surface descriptions of the CAD models. The CAD models used as the underlying description tool should be capable of providing these descriptions.

Spatial relationships between statica objects can be represented by homogeneous transformations between their body attached frames. These relationships can be variable or fixed. Variable relationships allow one statica member to move relative to another. The permissible motions are described by the relationship between the two objects and the restrictions that apply to the relationship. Variable relationships between two statica objects do not result in other statica objects. Variable relationships between statica model objects which have relative motions between their sub-structures. Fixed relationships do not allow one statica member to move relative to another.

The operation that maintains variable relationships between statica objects is <CONNECT> over homogeneous matrices. Homogeneous matrices are aggregation parameters describing the relationships. <COMBINE> maintains fixed relationships between statica objects. These are the only two operators required to show the relationships between two objects with concrete properties.

Examples of statica instances from the MIL manufacturing workcell illustrate the information employed to define those instances. Instantiation of the object models is not part of the MEDL language. User interface routines like data entry screens have to be written to support instantiations of objects declared in MEDL.

MEDL object models can be used to drive a variety of user interaction mechanisms, such as solids geometric modelling tools, data entry screens, etc., to create data for the object instances. Some of the modelling tools also permit derivation of properties, like areas, volumes, etc., from the object descriptions. The information required to model statica is MEDLI installation dependent. The corresponding MEDL terminal list for "concrete_properties" will specify this. In the following examples, assume, for simplicity, that the property color can be defined by only one color for every object.

Statica Examples

```
1. Aircraft_Part_1 : statica ;
  { The properties shown below are to be defined during
    instantiation of the statica object Aircraft_Part_1

   concrete_properties
     geometry: SHAPE: shown in Figure 3.4(a)
                 .
                 .
               BODY_ATTACHED_FRAME:
                      shown in Figure 3.4(a)
     WEIGHT   : 0.25 kg
         .
         .
     COLOR    : GREEN
     MATERIAL: ALUMINUM                                }

2. Aircraft_Part_2 : statica;
  { The properties shown below are to be defined during
    instantiation of the statica object Aircraft_Part_2
```

```
        concrete_properties
          geometry: SHAPE: shown in Figure 3.4(b)
                          .
                          .
                         BODY_ATTACHED_FRAME:
                                shown in Figure 3.4(b)
          WEIGHT   : 0.3 kg
              .
              .
          COLOR    : GREEN
          MATERIAL: ALUMINUM
```

Other statica instances from the MIL example workcell are Aircraft_Part_3 shown in Figure 3.4(c), Aircraft_Part_4 shown in Figure 3.4(d), a Conveyor_Belt and Conveyor_Base, a Flange and a Flange_Holder_Base.

Informata

Informata models the informational content of a manufacturing workcell. Informata is the information generated by sensors in the workcell and modified by the workcell processes. Informata is used for modelling the perceived world model in the workcells. Since informata instances are data, the instances can be modelled and maintained using databases and object-oriented data structures.

Informata Definition

informata ::= abstract_properties

The terminal symbols used to define abstract_properties, are left undefined. The implementation language and operating system of the target modelling facility will provide the data structures of their choice for defining the informata_attributes. For example, PASCAL data structures can be used for informata_attributes.

Informata Examples

```
1. list_of_object_areas : informata ;
  { The data structures shown below are to be defined
    during instantiation of the informata object
    list_of_object_areas

   abstract_properties: ARRAY [1..MAXOBJECTS] of real; }

2. list_of_centroids : informata ;
  { The data structures shown below are to be defined
    during instantiation of the informata object
    list_of_centroids

   abstract_properties:
                 record
                   Xc : ARRAY [1..MAXOBJECTS] of real;
                   Yc : ARRAY [1..MAXOBJECTS] of real;
                 end;                                 }
```

Data in databases can usually be maintained using three primitive operations: 1. <insert>, 2. <delete>, and 3. <update>. Regardless of the data management schema chosen for the implementation, these three primitive data operations will be permitted on informata. Informata will permit legal implementation language operations on the data structures defining them. The operators permitted on informata can themselves be modelled using processes. Processes are a basis object class to be defined later in this section and they provide the "action" aspects of abstract objects.

Sensor

Sensors act as transducers between the concrete and abstract domains. Sensors, therefore, have both physical and abstract attributes. Functionally, they cannot be modelled either as statica or informata due to their inherent transduction properties. For example, dynamic touch sensors consist of PVDF material which senses vibrations in the concrete domain and converts those vibrations to signature information in the abstract domain. The physical properties of the sensor define the physical attributes of the piece of PVDF. The transduction process is a function of the material and cannot be separated from it. The sensory output of the PVDF material is the abstract component of the touch sensor.

Sensor Definition

```
sensor ::= concrete_sensor   abstract_sensor
concrete_sensor ::= concrete_properties
abstract_sensor ::= abstract_properties
```

Sense inputs consist of the physical stimulations which result in sensation. Sense inputs need not be modelled, since they depend on the inherent physical properties of the sensory devices. The sense outputs are informata attributes which abstractly represent the results of physical simulations of the sensors. The sense outputs are modelled by the abstract_sensor. All sensors also have a physical form which is modelled by concrete_sensor. The concrete_sensor is used to model the physical world interactions of the modelled sensors.

The concrete and abstract properties of sensors are defined using concrete_properties and abstract_properties respectively. This allows concrete_sensor and abstract_sensor to inherit all the terminal lists created for concrete and abstract objects in MEDL. It also permits concrete_sensor to be operated by all statica operators and abstract_sensor to be operated by all informata operators. Sensors can be treated both as statica and informata depending on the operating application functions. For example, in performing collision avoidance, the physical properties of the sensors such as the space it occupies is important. But, when processing sensory outputs in a controller, the informational content of the sensors is important and not its physical properties.

Sensor Examples

```
1. PVDF_touch_sensor: sensor ;
  { The concrete properties of the PVDF sensor and the
    abstract properties of the sensor output are
    instantiated as shown below

   concrete_sensor:
             geometry: SHAPE: Thin-Film
                             .
                             .
             COLOR   : WHITE
             MATERIAL: PVDF
   abstract_sensor: time_signature_of_touch_sensor      }

2. vision_camera: sensor ;
  { The concrete properties of the vision camera and the
    abstract properties of the sensor output are
    instantiated as shown below
```

```
concrete_sensor:
          geometry: SHAPE: cuboid
                            .
                            .
          COLOR    : BLACK
abstract_sensor: grey_scale_image_of_scene
```

Actuators

Actuators are transducers between the abstract and concrete domains. They have both descriptive and action parts. Actuators take informata as inputs and produce physical changes that can be recognized in the concrete domain. The resultant actuation usually produces motion in the physical world. For example, given an input set-point, a d.c. servo will move its armature to reach that set-point. This motion results in physical world changes such as moving a shaft or a conveyor connected to the shaft, etc.

Actuator Definition

```
actuator ::= concrete_actuator   abstract_actuator
concrete_actuator ::= actuator_description
                                       actuator_action
actuator_description ::= concrete_properties
actuator_action ::= LINEAR_MOTION | ANGULAR_MOTION
abstract_actuator ::= abstract_properties
```

As the above definitions show, actuators perform the opposite transduction of that performed by sensors. The concrete_actuator defines the form of the actuators and the physical actions of the actuators. The actuator_actions are defined in terms of the homogeneous matrix variables which describe the variable relationships between two statica objects. All operations allowed on statica objects are permissible on the concrete_actuator as well. So, actuators can be connected or joined to other statica or objects with concrete_properties. Likewise, the abstract_actuator is the definition of the abstract properties of actuators such as the control parameters. The abstract_actuator allows the modelling of the control and actuation inputs of the actuators. All operations permitted on the informata objects are permissible on actuator_inputs.

Actuator Examples

```
conveyor_dc_servo: actuator ;
{ The concrete properties of the conveyor_dc_servo
  actuator, its action and its abstract inputs are
  instantiated as shown below

actuator_description:
       geometry: SHAPE: cylindrical
                          .
                          .
       WEIGHT  :  1.3 kg
         .
       TEXTURE : POLISHED_STEEL
actuation_action: ANGULAR_MOTION
                     (described by the homogeneous
                      matrix variable parameter)
abstract_actuator: ON_OFF                              }
```

Actuators are permitted an additional operator applicable to the entire actuator object. The <DRIVE> operator is permissible when it can propel two objects with statica attributes. For the actuator to perform actuation, it requires an object of reference relative to which it can propel another object. The two objects should also have a permissible variable relationship between their statica attributes. For example, for a motor to drive a robot link relative to another, the stator of the motor should be connected to the reference link and the rotor to the propelled link. Both links should be "connected" to each other through a joint. The variable relationship between the two links can be described using a homogeneous matrix. The actuator_action "drives" the variable parameter of the homogeneous matrix of the connection to change the relationship according to abstract_actuator inputs.

Processes

Processes are purely abstract action entities. Processes, therefore, only have an action part. Processes take informata inputs and transform them into informata outputs. Inputs for processes could come either from informata, abstract_sensor outputs or other process_outputs. Outputs from processes go either to informata, abstract_actuator inputs or process inputs. Figure 3.5 shows a detailed diagram of the process data interactions. Process objects are described as shown below:

Process Definition

```
process ::= process_input  process_plan  process_output
process_input ::= abstract_properties
```

```
process_output ::= abstract_properties

process_plan ::=   servo_level_plan

                 ¦ command_level_plan

                 ¦ task_level_plan

                 ¦ planning_level_plan

                 ¦ goal_level_plan
```

The level of process control depends on the level of the process_plan. As an illustration of command_level_plans, we consider process examples from the Optomation vision system. Three example commands recognized by the optomation VPL command language are shown in the example below.

Process Examples

```
{ Shown below are the three instantiations for
  optomation_area_computation,
  optomation_centroid.x_computation,
  optomation_centroid.y_computation declared
  as processes                                        }

1. optomation_area_computation: process ;

{  process_input: binary_image_of_scene
   process_output: list_of_object_areas[item#]
   process_plan: AREA(item#)                          }

2. optomation_centroid.x_computation: process ;

{  process_input: binary_image_of_scene
   process_output: list_of_centroids.Xc[item#]
   process_plan: CENTROID.X(item#)                    }

3. optomation_centroid.y_computation: process ;

{  process_input: binary_image_of_scene
   process_output: list_of_centroids.Yc[item#]
   process_plan: CENTROID.Y(item#)                    }
```

Processes have abstract_properties for their inputs and outputs. All operations applicable on abstract_properties are permitted on process inputs and outputs. Processes also have an operator <TRANSFORM> which allows processes to be applied to modify informata. Processes can be applied to controlled objects in the manufacturing workcell. As mentioned in Chapter Two and in [Pal and Doty 1987], process_plans can be at different hierarchical levels depending on the aggregation levels of the objects controlled with the process.

Classification of a Functional Basis

In this section, so far, we have described the properties of five objects--statica, informata, sensors, actuators and processes--used as a basis for describing object instances and object models in MEDL. The distinctions are based primarily on the concrete or abstract domain in which the objects fall. Concrete operations are based on the specification of spatial relationships between concrete objects. Likewise, common properties classify the abstract domain as well. We separate the functional basis along their concrete and abstract properties. This allows us to use common operations to describe aggregations between objects of common physical and abstract properties. The basis properties are categorized into concrete and abstract as follows,

Description of Basis Domains

```
concrete_basis ::=   statica
                   | concrete_sensor
                   | actuator_description
abstract_basis ::=   informata
                   | abstract_sensor
                   | abstract_actuator
                   | process
```

3.5 Aggregations

Aggregations provide MEDL users mechanisms for setting up object models for manufacturing workcell components. Aggregations in MEDL are similar to user-defined data structures permitted in some higher-level computer programming languages such as PASCAL, C, etc. Aggregation definitions are based on the basis objects or other pre-defined aggregations. Aggregation_rules define the interactions between the aggregation members.

All aggregations in MEDL must contain either concrete descriptions, or abstract descriptions of the object or both. Aggregations used as MEDL object types describe the properties of aggregate_names. These properties can be used to define instances in the manufacturing databases. Aggregations have the syntax,

Description of MEDL Aggregations

```
aggregation ::= aggregate_name 'DESCRIBED_BY'
                                     aggregation_rule ';'

aggregate_name ::= identifier

aggregation_rule ::=   concrete_declaration '&' <NULL>

       | <NULL>                 '&' abstract_declaration

       | concrete_declaration '&' abstract_declaration
```

An example for aggregations is the declaration of a general type of object controller. Controller in our example is defined by its body as the concrete part and the command interpreter as the its abstract part. In the MIL workcell, there are two instances of controllers. A high level description of these controllers and an instantiation of the controller objects are

Examples of MEDL Aggregations

```
controller DESCRIBED_BY
  controller_body DEFINED_BY     .
                                 .
&
  command_interpreter DEFINED_BY     .
                                     . ;

optomation_vpl_controller,
        ge_p50_controller : controller ;
```

Concrete Declarations of Aggregations

Concrete_declarations specify the concrete properties of objects. All concrete objects and concrete parts of an object are described by their body attached frames. A minimal description of concrete objects should at least have a statica description which has a body attached frame. This body attached frame is used to reference the object to other concrete objects. Concrete descriptions of objects also have an optional concrete_object portion. The concrete_object part is a description of the physical part of the aggregation as relationships between its constituent object models.

MEDL Concrete Part Declaration Syntax

```
concrete_declaration ::=
  (CONCRETE_PART)
     concrete_name DEFINED_BY concrete_basis
                ( AND concrete_object )

concrete_name ::= identifier

concrete_object ::=   concrete_connections
          ¦ concrete_object  concrete_connections

concrete_connections ::=
        AND <COMBINE> concrete_name
                AT [homogeneous matrix]
      ¦ AND <CONNECT> concrete_name
                AT [homogeneous matrix]
      ¦ AND <DRIVE> <CONNECT> concrete_name
                AT [homogeneous matrix]
                USING actuator_action
      ¦ AND <COMBINE> concrete_basis
                AT [homogeneous matrix]
      ¦ AND <CONNECT> concrete_basis
                AT [homogeneous matrix]
      ¦ AND <DRIVE> <CONNECT> concrete_basis
                AT [homogeneous matrix]
                USING actuator_action
```

<COMBINE> and <CONNECT> are the only two operators required for describing relationships between the concrete properties of objects in a manufacturing workcell. <COMBINE> describes fixed relationships between body attached frames

of concrete objects. <CONNECT> describes a variable relationship between concrete object body attached frames. Spatial relationships are described relative to a reference frame. In MEDL, the body attached frames of concrete objects on the left hand side of the <CONNECT> and <COMBINE> operators provide the reference frames for the operation. The reference body attached frame of the leftmost object stays as the reference frame for the compound object created by the <CONNECT> and <COMBINE> operators.

Homogeneous matrices can be used as the aggregation parameters for describing the relationships. Homogeneous matrices are (4 X 4) matrices with a (3 X 3) rotational part and a (3 X 1) translational part which describe the displacement and orientational relationships between two body attached frames. In a <CONNECT> operation, the homogeneous matrices have variable rotational or prismatic parameters, which permit the variable relationship between their body attached frames to be modelled. The concrete objects form a kinema object through this <CONNECT>ion.

The operator <DRIVE> is a special actuator_action. Actuators can move concrete objects relative to one other only when motion is permitted between them. The permissibility of relative motion is indicated by the <CONNECT> operation between two concrete objects. The <CONNECT>ed objects can therefore be <DRIVE>n by the actuator_action. The actuator_action specifies the nature of change of the <CONNECT>ion variable. For example, a simple rotational actuator will change only the rotational part of the homogeneous matrix, while a linear actuator will change only the translational part. The changing parameter can be properly attached to the homogeneous matrix, thus providing a control of the nature of the variable relationship. The two concrete objects connected through an actuator form an actuated kinema.

In our previous example of a controller, the controller_body is defined by the physical attributes of the controller box. Included in this description are the concrete properties of the controller body and its body attached frame. This concrete part does not have any combinations of other objects. Later examples of robot and workcell aggregations detail other aggregation rules.

Example for Concrete Declarations

```
controller_body DEFINED_BY statica
```

Abstract Declarations of Aggregations

Abstract aggregations are collections of the abstract properties of objects. Similar to concrete declarations of object models, the abstract declarations are also dependent on the abstract object declarations as shown below:

MEDL Abstract Part Declaration Syntax

```
abstract_declaration ::= abstract_name DEFINED_BY
                                        abstract_object
```

```
abstract_name ::= identifier

abstract_object ::=   abstract_basis

                ¦ transformation

                ¦ abstract_name

                ¦ abstract_object  AND  abstract_basis

                ¦ abstract_object  AND  transformation

                ¦ abstract_object  AND  abstract_name

transformation ::= <TRANSFORM> src_of_abstract_inputs  TO
                               dest_of_abstract_outputs
                   USING process

src_of_abstract_inputs ::=   abstract_sensor

                           ¦ informata

                           ¦ process_outputs

dest_of_abstract_outputs ::=   abstract_actuator

                             ¦ informata

                             ¦ process_input
```

<TRANSFORM> is an abstract operator which produces outputs as a function of its inputs and the state of the process used. outputs. The inputs for the <TRANSFORM>ation can be derived from any source of informata--sensory outputs, informata, or outputs from other processes. The outputs from <TRANSFORM>ations are applied to actuator inputs, informata or inputs for other processes. The agents for <TRANSFORM>ations are the abstract action entities, processes. Using this mechanism, different processes can be chained to generate data of higher levels of abstraction from lower-level data. This model of abstraction is

applicable to the different levels of the perceived world model.

The abstract part declaration for the controller example will be,

Example for Abstract Part Declaration

```
command_interpreter DEFINED_BY process
                    AND        process
                    AND        process
                                  .
                                  .
                    AND        process ;
```

Suppose the controller abstract part was declared using N processes. This controller object type can be used to model controller instances with up to N commands. The unused commands can be set to default to no-operations (no-op's). The command interpreters for controller instances will be fully described when process instantiations specify the executable commands.

The full description of our controller aggregation therefore looks like,

Example for MEDL Aggregations

```
controller DESCRIBED_BY controller_body
  CONCRETE_PART
      controller_body DEFINED_BY statica
&
  ABSTRACT_PART
      command_interpreter DEFINED_BY process
                          AND  process
                          AND  process
                                  .
                                  .
                          AND  process ;
```

We use the optomation vision controller in the MIL manufacturing workcell as an example instantiation for a controller. The following are a few parameters which describe the optomation_vpl_controller.

Controller Example

```
optomation_vpl_controller : controller;

{ This example shows the instantiation of an
  optomation vision controller

controller_body:
    geometry: SHAPE: cuboid
                   .
                   .
    WEIGHT  : 100 kg
        .
        .
    COLOR   : BLACK
    MATERIAL: STEEL_CASE

command_interpreter:
     optomation_area_computation
     optomation_centroidx_computation
     optomation_centroidy_computation
                    .
                    .                      }
```

In this section, we provide a few examples for aggregations used to define some common classes of components found in a manufacturing workcell. A manufacturing workcell in the MIL will be used as a source for our examples. Later in this section MEDL is used to describe the entire MIL workcell.

Examples for Aggregations

We start our examples by building a special purpose object model from the MIL manufacturing workcell. The aggregation shown below describes a mobile perceptor. The perceptor consists of a sensory device and a controller. The perceptor is modelled by a statica, which, in this case, is simply a body attached frame. To the body attached frame description, <COMBINE> the controller_body that houses the informata action part of the controller and processes the sensory output to provide perception. The concrete_sensor describes the physical attributes of the sensor and a variable <CONNECT> operation describes its position with respect to the perceptor body attached frame. The condition of the homogeneous matrix at any given time indicates the pose of the sensor body attached frame relative to the perceptor base frame.

The abstract part of a perceptor consists of two stages. In the first stage, the abstract_sensor, the sensory output data, is <TRANSFORM>ed into an informata data structure using a sensory process. In the second stage, the command interpreter of the controller uses informata input to derive higher level details of the sensed data.

Mobile Perceptor Aggregation Declaration

```
mobile_perceptor DESCRIBED_BY
  CONCRETE_PART
     perceptor_body DEFINED_BY statica
                    AND <COMBINE> controller_body
                            AT [homogeneous matrix]
                    AND <CONNECT> concrete_sensor
                            AT [homogeneous matrix]
&
  ABSTRACT_PART
     perception DEFINED_BY
                <TRANSFORM> abstract_sensor TO
                           informata  USING process
                command_interpreter ;
```

There are two examples for mobile_perceptors in the MIL workcell. The first is a vision system and the second a touch system. Both, the PVDF touch sensor as well as the vision camera are mounted at the end-effector of a P-50 robot manipulator. The optomation vision controller and the touch controller are located outside the robot work envelope. The motion of the sensors relative to the respective controller frames can be specified using homogeneous transformations and are related to the configurations of the robot. The vision system has a pre-processor which converts the raw grey scale image into a binary image. The vision system command interpreter uses this binary image for its computations. The touch sensor also has a pre-processor which takes the electrical analog signal and digitizes it for the higher level algorithms. The following are example instances for the two MIL mobile perceptors.

Mobile Perceptor Examples

```
optomation_vision_system,
           touch_system : mobile_perceptor ;

{  High-level instantiation example of the mobile
   perceptor optomation vision system. The detailed
   instantiation would describe the statica properties
   in detail as in our previous examples, etc.

optomation_vision_system DEFINED_BY statica
                               {a body attached frame}
        AND <COMBINE> optomation_controller
                         AT [homogeneous matrix]
        AND <CONNECT> vision_camera
                            AT [homogeneous matrix]

optomation_vision_processing DEFINED_BY
    <TRANSFORM> vision_output TO
         binary_image  USING vision_pre_processor ;  }
```

Our next aggregation example are general, 6 degrees of freedom (dof) and 5 dof robot manipulators. These aggregations define the robots with respect to their base. The transformations along the various links of the manipulators describe the variations of the link frame with respect to the base frame of the robot manipulator. The product of these transformations out to the end-effector computes the forward transformation of the manipulator in cartesian coordinates with respect to the base frame of the robot. In the aggregations given below, link_1, link_2, etc., are defined as type statica and carry concrete properties which define the links of the robots and its kinematic structure. Instantiations of the robot object models should specify the concrete properties of the various links and the different A matrices which are homogeneous matrices describing the connections.

<u>6-DOF and 5-DOF Manipulator Aggregation Declarations</u>

```
robot_6dof DESCRIBED_BY
 CONCRETE_PART
  base_frame_6 DEFINED_BY statica
               AND <CONNECT> statica AT [0A1] {link_1}
               AND <CONNECT> statica AT [0A2] {link_2}
               AND <CONNECT> statica AT [0A3] {link_3}
               AND <CONNECT> statica AT [0A4] {link_4}
               AND <CONNECT> statica AT [0A5] {link_5}
               AND <CONNECT> statica AT [0A6] {link_6}
& ;

puma_260, cm_t3 : robot_6dof ;

robot_5dof DESCRIBED_BY
 CONCRETE_PART
  base_frame_5 DEFINED_BY statica
               AND <CONNECT> statica AT [0A1] {link_1}
               AND <CONNECT> statica AT [0A2] {link_2}
               AND <CONNECT> statica AT [0A3] {link_3}
               AND <CONNECT> statica AT [0A4] {link_4}
               AND <CONNECT> statica AT [0A5] {link_5}
& ;

ge_p50_robot : robot_5dof ;
```

Workcells permit a recursive definition based on other predefined workcells. Other basis and aggregate objects can also be combined with the workcells. Workcells are defined by their workcell_base_frames. Workcell objects, their spatial relationships and their functions define the types of tasks permissible in the workcells. Consider the building of a generic workcell which permits one 5 dof robot, two mobile perceptors, a controller and five statica objects in it. This is close to the MIL workcell description. Any generic workcell which can fit under this description can be instantiated using this model. If some of the resources like two statica out of the five allowed are not present, the instantiation of the workcell will set the corresponding object parameters to <NULL>. If the workcell model is inadequate for the workcell instance being defined, then a different model should be used or defined. If the user specifies an inadequate or incomplete description of the workcell, he/she knowingly assumes the risk of inaccurate modelling of the workcell.

Aggregation Declaration for Workcell

```
workcell DESCRIBED_BY
 CONCRETE_PART
  workcell_concrete DEFINED_BY statica
                                {workcell base frame}
                    AND <COMBINE> statica
                               AT [homogeneous matrix 1]
                                      .
                                      .
                    AND <COMBINE> statica
                               AT [homogeneous matrix 5]
                    AND <COMBINE> perceptor_body
                               AT [perceptor homo matrix 1]
                    AND <COMBINE> perceptor_body
                               AT [perceptor homo matrix 2]
                    AND <COMBINE> controller_body
                               AT [controller homo matrix]
                    AND <COMBINE> base_frame_5
                               AT [robot homo matrix]
&
 ABSTRACT_PART
  workcell_abstract DEFINED_BY perception
                    AND        perception
                    AND        command_interpreter ;
```

The previous basis and aggregate examples modelled devices from the MIL manufacturing workcell. This example combines all the devices and use them for an example instance of the aggregation workcell. The MIL workcell contains a GE P50 robot, a robot controller, a touch system, a vision system, and three tables forming a protective barrier around the workcell. These devices combined describe the MIL workcell as an instance.

There are two parts to the workcell description. One is the physical layout of the workcell and is modelled by the physical or concrete attributes of the workcell devices. The second part is the logical connectivity of the workcell. This is inherent in the device logical connections made in the device aggregation descriptions and instantiation. This comes out as the abstract description of the workcell.

```
mil_workcell : workcell ;

{ Instantiation of the mil_workcell as an example
  of the aggregation workcell
mil_workcell_concrete DEFINED_BY statica
                  { body attached frame of reference }
                AND <COMBINE> table_1 AT [offset]
                AND <COMBINE> table_2 AT [offset]
                AND <COMBINE> table_3 AT [offset]
                AND <COMBINE>       <NULL>
                AND <COMBINE>       <NULL>
                AND <COMBINE> optomation_vpl_controller
                                 AT [offset]
                AND <COMBINE> touch_system AT [offset]
                AND <COMBINE> ge_p50_controller AT [offset]
                AND <COMBINE> ge_p50_robot AT [offset]
&
mil_workcell_abstract DEFINED_BY optomation_vision_system
                AND        touch_system
                AND        ge_p50_controller ;           }
```

where table_1, table_2, table_3 : statica ;

The plant_floor is defined as a combination of other plant_floors and workcells. The plant_floor is defined by the PLANT_FLOOR_BASE_FRAME. This definition permits the plant floors to be later integrated into the plant or facility description. The PLANT_FLOOR_FRAME acts as the universal base frame for the plant_floor and all the workcells in plant_floor can be defined with respect to this base frame. Suppose a plant_floor is modelled to permit up to a maximum of 10 workcells on the floor, the description of the aggregation for plant_floor would appear as shown below.

<u>Aggregation Declaration for Plant Floor</u>

```
plant_floor DESCRIBED_BY
 CONCRETE_PART
  plant_floor_concrete DEFINED_BY statica
                                 { body attached frame }
                      AND <COMBINE> workcell
                                  AT [homogeneous matrix 1]
                                     .
                                     .
                      AND <COMBINE> workcell
                                 AT [homogeneous matrix 10]
&
 ABSTRACT_PART
  plant_floor_abstract DEFINED_BY workcell_abstract
                                           .
                                           .
                      AND        workcell_abstract ;
```

where the homogeneous matrix defines the fixed relationship between plant_floors and workcells which are combined to form the new plant_floor.

In the aggregation for plant, plant is defined by its PLANT_BASE_FRAME. If the plant is chosen as a base or universal frame, its base_frame will be identity. The functions of the plant and the object descriptions of the plant are derivable from objects which are used in forming the plant.

Aggregation Declaration for Manufacturing Plant

```
plant DESCRIBED_BY
 CONCRETE_PART
  plant_concrete DEFINED_BY statica
                                 {body attached frame}
                 AND <COMBINE> plant_floor_concrete
                               AT [homogeneous matrix]
                                 .
                                 .
&
 ABSTRACT_PART
  plant_abstract DEFINED_BY plant_floor_abstract
                                 .
                                 .
```

These examples for aggregations progress all the way from the device level on to the plant level. The plans of actions at the different levels will correspond to their level of modelling abstraction. At the basis process, control will be at the servo and command levels. In workcells, plant_floors and plants, the control will be at the task, planning and goal levels respectively. All these levels of control can be modelled using processes and controllers executing different levels of process plans. We will pursue tasking primitives further in Chapter Four.

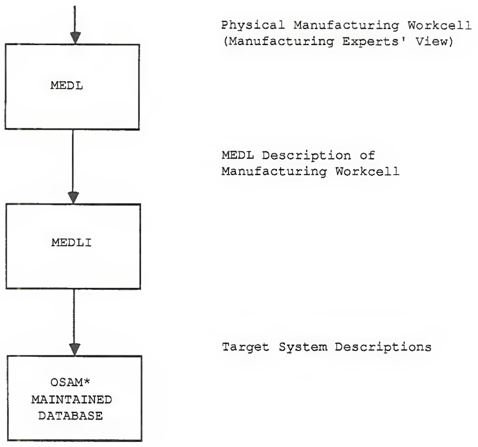

Physical Manufacturing Workcell
(Manufacturing Experts' View)
MEDL
MEDL Description of
Manufacturing Workcell
MEDLI
Target System Descriptions
OSAM*
MAINTAINED
DATABASE

FIGURE 3.1 Functions of MEDL and MEDLI

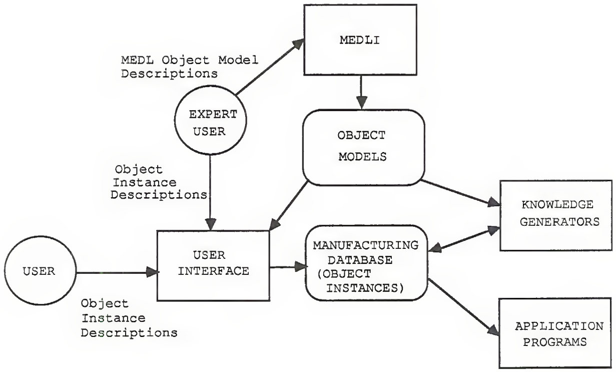

MEDLI
MEDL Object Model Descriptions
EXPERT USER
OBJECT MODELS
Object Instance Descriptions
KNOWLEDGE GENERATORS
USER
USER INTERFACE
MANUFACTURING DATABASE (OBJECT INSTANCES)
Object Instance Descriptions
APPLICATION PROGRAMS

FIGURE 3.2 MEDL and MEDLI Applications

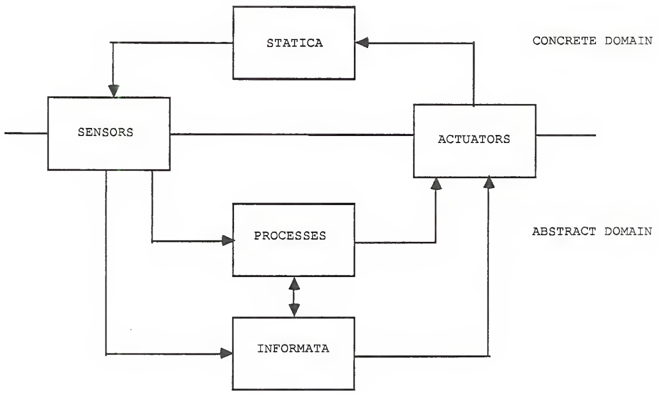
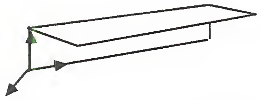
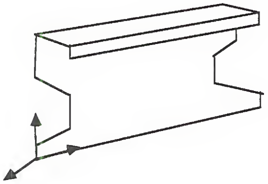


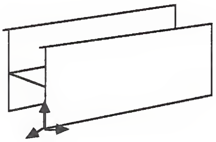
FIGURE 3.3 MEDL Functional Basis



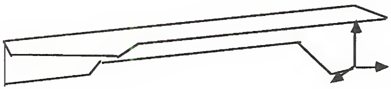
(a) Aircraft Part 1



(b) Aircraft Part 2



(c) Aircraft Part 3



(d) Aircraft Part 4

FIGURE 3.4 Statica in MIL Manufacturing Workcell

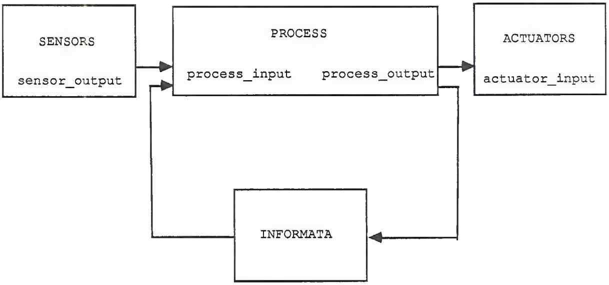


FIGURE 3.5 Process Data Interactions

CHAPTER FOUR

TASKING PRIMITIVES FOR MOTION PLANNING IN COOPERATING MULTI-ROBOT WORKCELLS

There is a lack of standardization prevalent in the robot industry today. Multiple types of robot structures exist for robot manipulators. There are just as many different robot controllers and control languages available as there are robot manufacturers. This lack of standardization poses problems, particularly for the customers who have to use robot manipulators from multiple vendors and the floor personnel who must handle and maintain the robot software and hardware equipment. General purpose tools that help overcome this lack of standardization in manufacturing equipment are being developed [Govindaraj and Doty 1986].

Manufacturing workcells are becoming more complex as multiple intelligent devices are integrated into the workcell and coordinated by intelligent cell controllers. The workcells are treated as islands and are tied together using factory floor and plant level networks. It is foreseen that many of the future manufacturing workcells will have multiple cooperating manipulators [NSF 1987].

The Machine Intelligence Laboratory at the University of Florida has been focusing its attention on cooperating robots for some time now [Doty 1983, Govindaraj and Doty 1985, Chand 1984]. As a result, the motion tasking primitives developed in this paper treat the general case of multiple cooperating manipulator workcells and identifies tasking primitives for cooperating robots. However, a taxonomy for robot cooperation will precede development of cooperating robot tasking primitives in order to establish the range of applications of those primitives.

Before discussing robot cooperation and tasking primitives, a couple of points about our effort need to be mentioned. Much of the prevalent techniques in robot programming involve single robots. Typically, teach pendants are used to specify and record end-effector poses in planning trajectories and specifying tasks. In the case of multiple manipulators it becomes confusing, if not impossible, to specify the task in end-effector poses. Since the principle objective in motion planning in multiple manipulators is the transfer or manipulation of workpieces, we decided to specify tasks based on the workpiece poses and trajectories rather than the manipulator end-effector poses. We call this workpiece oriented programming. In the literature the same concept has been referred to as object oriented programming [Faverjon 1987]. In deference to the ambiguity of the term and its extensive use in database systems and programming languages, we prefer "workpiece" to "object." The other point that should be made here is that in our tasking primitives, collision detection and avoidance is ignored. The assumption we make is that a watchdog collision detection and avoidance system will be available in the multi-robot workcell to identify potential collisions [Kilmer et al. 1984].

4.1 Cooperating Robots

Manipulators cooperate when more than one manipulator is involved in the task being performed and data interaction between the manipulators is required for the successful completion of the desired task. Currently, researchers are studying the problem of manipulator cooperation in two broad classes: (1) cooperation in legged mobility and (2) cooperation in workpiece manipulation. Some of the principles developed in each class may be applicable to the other, but the difference in emphasis often leads to different aspects of cooperation.

In legged mobility cooperation, multiple kinematic linkages act as legs in producing the motion of a mobile platform. Investigation into such legged locomotion is receiving much attention and ranges from one-legged hopping machines [Raibert et al. 1984] to six-legged hexapods [Orin 1976]. Experimental legged vehicles of many different configurations have been built in the past and are discussed in [McGhee 1984]. In legged robots, the research is centered around gait analysis, balance of the mechanisms, moving a platform using multiple kinematic linkages and adaptability to rough terrains.

Cooperation in workpiece manipulation has not received quite as much attention as cooperation in locomotion. In workpiece manipulation, one can envision two kinds of cooperation: (1) Cooperation between the multi-jointed fingers of multi-fingered hands [Salisbury and Craig 1982, Jacobsen et al. 1984] and (2) Cooperation between multiple manipulator arms with their various end-effectors. Although both configurations share the same abstract conceptual base of workpiece manipulation with multiple, serial linkages, they differ in emphasis.

In multi-fingered hands, the emphasis is on point contact grasping of workpieces with the finger tips and the balancing of the workpieces with this point contact grasping. The range of finger motion is restricted to small volumes and workpiece manipulation is restricted to cooperative manipulation only. Individual fingers are usually incapable of manipulating the workpieces independently due to the point contact nature of the grasping.

Since cooperating multiple manipulator configurations can be perceived as a large hand whose manipulator arms serve as the multi-jointed fingers of a monstrous hand, cooperation between multiple manipulator arms deals with all the above considerations and introduce a few more interesting aspects of their own. Manipulators, in contrast to fingers, support various kinds of end-effectors and typically are not restricted to point contact grasping.

Balancing the workpieces should involve sharing the workpiece load between the cooperating manipulators. Due to the large size of the workspaces, both large and small motions are involved. Individual manipulators are capable of handling the workpieces independently, as well as cooperatively with the other manipulators. Therefore, tasking for this class of robot cooperation involves both individual and cooperative task planning.

We will restrict our discussions to cooperation in workpiece manipulation among multiple manipulator arms.

4.2 Robot Cooperation in Multi-Robot Manufacturing Workcells

Based on the types of cooperation foreseen in manufacturing workcells, cooperating tasks fall into four classes: 1. Transfers, 2. Intelligent Fixturing, 3. Machining, and 4. Assembly.

Transfer operations move workpieces from a source pose to a destination pose. The two poses may or may not be in the same manipulator workspace. Transfer actions require cooperation when more than one manipulator is required to move the workpiece. Static and kinetic interactions are possible between the cooperating manipulators during transfers.

One or more robot manipulators may potentially serve as programmable, reconfigurable intelligent fixtures to assist other manipulators performing operations on the workpieces. While such robots act as fixtures for the workpiece, other cooperating manipulators perform machining or assembly operations on the robot-fixtured workpieces. At present, robot manipulators may not be capable of providing stable fixtures for high precision operations, but should be adequate for low tolerance applications such as snap-fit assembly, non-contact welding, etc. In spite of the unsuitability of intelligent fixturing for high precision machining operations with current technology, the concept of cooperating intelligent fixtures appears to be worth pursuing.

Machining operations require cooperation when manipulators are used as intelligent fixtures for other manipulators to perform the specified machining operation. During machining, only dynamic interactions are possible between the manipulators. Both the manipulators must be capable of exerting opposing forces to effect the machining operation.

Assembly operations will probably be the most common application for multi-robot cooperation. The cooperating manipulators might simultaneously assemble the workpieces on a common spindle or perform intelligent fixturing for each other during the assembly and mating of parts.

Machining and assembly operations are really compound operations which may be further analyzed into single-robot primitives or simpler cooperating-robot primitives. Task analysis and proper sequencing by the planner are required for successful machining or assembly operations.

4.3 Task Planner for a Cooperating Multi-Robot Workcell

A multiple layer hierarchical task planning system is being developed in the Machine Intelligence Laboratory, University of Florida [Pal and Doty 1987]. The architecture of the cooperating multi-robot task planner is presented in Figure 4.1. This task planner is a part of the Manufacturing Workcell Design and Tasking Facility also under development. The task planner uses Tasking Primitives, Manufacturing Procedures, and Task Decomposition Rules in recursively analyzing and decomposing tasks specified at a higher level into lower levels of detail. The planner is being designed to work at five levels of task abstractions, (1) the Production Control Level, (2) the Manufacturing Process Level, (3) the Workcell Level, (4) the Manipulator Level and (5) the Servo Level. In the top three layers, the tasks do not have any component dependencies and can execute on any arbitrary workcell whose model is available to the planner. The tasking primitives being developed should be device independent and provide adequate building blocks for specifying higher level tasks based on them. The primitives should also provide the device specific interface for the planner through translators into the device control languages.

MEDL processes can be used to model task specifications at the various tasking levels. The tasking primitives described in this chapter can be modelled using workcell level processes.

4.4 Tasking Primitives

The proliferation of robot control languages makes the users' task difficult by having to train personnel with as many languages as their plant uses. Most of the robot languages are written for controlling single robot manipulators. Even the languages that permit multi-robot operation in their design do not provide any meaningful constructs for multi-robot cooperation tasks. Robot language designers only make passing remarks about the need for multi-robot constructs in robot languages [Ruoff 1979, Lieberman and Wesley 1977, Finkel et al. 1974].

In the Machine Intelligence Laboratory, during the development of specifications for multi-robot cooperation, we had the choice of developing a language for multi-robot cooperation or a robot language independent facility for non-procedural specification of cooperating robot tasks. Instead of designing another robot language which may later prove to be inadequate, we decided to design a facility for programming cooperating robots. The facility should also allow different types of sensors and other manufacturing workcell components to be programmed towards achieving the required cooperation task.

Two requirements for the tasking facility are that it must,

(1) Specify and plan tasks at multiple levels of abstractions [Pal and Doty 1987], and

(2) Make the task specifications independent of the constraints and details of the device programming languages.

These two requirements necessitate the development of sufficient, device language independent constructs for defining tasks at a higher level than the device level. These constructs, called Tasking Primitives, act as macros for task descriptions and provide an interface between the devices at the lower level and the task planner at a high level. The tasking primitives hide the lower level device specifics from the planner while providing device interfaces for the planner to realize their higher level task plans. Specific, device language translators written for the primitives allow the general task plans to be executed on specific devices. The high-level, device independent task specifications will be recursively analyzed by an intelligent task planner until the task-plan tree contains only primitives at the leaf nodes. The translators between the primitives and device languages allow the planned tasks to be executed in the workcell devices.

We have identified three classes of task statements required to specify robot tasks: 1. Motion Primitives, 2. Perception Primitives, and 3. Control Primitives. Motion primitives permit the programming of motion commands for manipulators and other motion generating devices like conveyor belts. Perception primitives are used for the acquisition, processing and application of sensor data. Control primitives are used to control motion and sensor devices in the workcells. The following example illustrates these three classes of primitives.

Suppose the specified task is to use a manipulator with a parallel jaw gripper to approach a workpiece and pick it up by exerting a specified force on the workpiece. This task can be broken down into the following sequence of commands:

1. Motion commands to move the manipulator end-effector from its current configuration to the workpiece pose

2. Motion commands to close the gripper

3. Perception commands to monitor the force sensors on the gripper

4. Control commands to monitor the force sensor until the specified force threshold is reached and stop closing the gripper.

These sequence of commands can operate in three parallel streams of motion, perception and control commands. The synchronizations between these three streams will determine the realization of the task for which they were planned. This paper defines the primitives required for motion task specifications in cooperating robot workcells. Future work will focus on Perception and Control primitives. The motion task primitives include single robot and cooperating multi-robot primitives.

4.5 A Taxonomy for Robot Motion Tasking Primitives

In developing the tasking primitives for robot motion planning tasks, it was necessary to model the tasks feasible in cooperating multi-robot workcells. One taxonomy of robot tasks is presented in Figure 4.2. The primitives for single robot and multi-robot motion tasking primitives will be the subject matter of the rest of this chapter.

At the first level of description, robot tasks can be broken down into single robot tasks and multi-robot tasks. Single robot tasks take place either in workcells where only a single robot is present or in multi-robot workcells where the task execution of one of the robots is completely independent of the tasks of the other robot. Although the two task classes are similar from the task planning and modelling perspective, they are different from the task execution point of view. During task execution and monitoring, single robot tasks in multi-manipulator workcells should be monitored more carefully for inter-manipulator collisions--particularly in the intersecting workspace volumes of the manipulators.

Following the taxonomy into the next lower level, the cooperation in workpiece manipulation can be adaptive or non-adaptive. Non-adaptive cooperation occurs when the cooperation between the participating manipulators takes place due to "canned" program sequences which cannot be altered as a result of the success or failure of the desired cooperating task. An example of a non-adaptive cooperation task occurs when two robots exchange workpieces without any signalling between them. As long as the robots are extremely accurate and do not miss a workpiece, task execution may continue to succeed. Loss of manipulator calibration, however, will result in failures of the cooperating tasks. Without sensory communication the manipulators will continue through their motions assuming that the operation was successful. The potential for such failures motivates the development of adaptive cooperation wherein data communication between cooperating manipulators becomes an important element of that cooperation. The cooperating robot tasking primitives developed here will only include the case of adaptive cooperation.

The next level in the taxonomy depends on the number of workpieces being manipulated by the multiple robots. Usually, single workpieces will be handled by cooperating robots for effecting transfers or machining. Manipulation of single workpieces can be further divided into two classes depending on the state of the workpiece during the cooperative manipulation.

For static manipulation, the workpiece pose is stationary during the rendezvous, while for kinetic manipulation the workpiece in motion during the rendezvous period. Both static and kinetic manipulation have two sub-classes where they can be controlled using either passive or active compliance between the manipulators. Static and kinetic manipulation and their sub-classes will be discussed in detail in later sections of this chapter.

Multiple workpieces are normally handled for assembly or are created by machining. Cooperative manipulation of multiple workpieces is at a higher level of abstraction than the two primitives just described. Such tasks can usually be decomposed into subtasks constructed from the previously defined primitives. At the Machine Intelligence Laboratory, we are currently investigating an AI based task planner for performing this task decomposition for assembly tasks in cooperative multi-robot workcells [Pal and Doty 1987].

4.6 Definitions Used for Modelling the Tasking Primitives

In this section, we define some of the concepts used later to develop motion tasking primitives.

In previous work on cooperation, Master-Slave systems have been defined and used [Nakano et al. 1974, Gavrilovic and Selic 1974]. The Master manipulator controls the workpiece trajectory while the other manipulator is a slave whose motion is dependent on the master manipulator. Since the roles of master and slave manipulators may need to change during a task, a fixed master-slave relationship will not be able to account for more sophisticated interaction. We introduce the more flexible, time-dependent concept of manipulator dominance. A parameter, coined the dominance factor, identifies the dominant and non-dominant manipulators at any instant of time during task execution.

Cooperating Workspace Volume (CWV)

The Cooperating Workspace Volume definition is required for establishing whether a cooperating task request is feasible. The CWV to allows simultaneous grasping of the workpiece by the manipulators participating in a cooperating task.

To define the CWV more clearly, first define the intersection workspace volume (IWV). The IWV is the volume created by the intersection of the workspace envelopes of adjacently located manipulators. The IWVs can be further divided into Mutual-Intersection-Workspace-Volumes and Universal-Intersection-Workspace-Volumes. A Mutual-Intersection-Workspace-Volume is formed by the workspace intersection of any subset of cooperating manipulators in the workcell, while the Universal-Intersection-Workspace-Volume, is formed by the workspace intersections of all the manipulators participating in the cooperating task. The task scheduler and planner, which analyzes and schedules cooperating tasks, should be aware of all the different IWVs for optimal planning and scheduling of cooperating tasks [Pal and Doty 1987].

The CWVs need not necessarily have intersecting volumes for robot cooperation to be possible. An example for this is the cooperative lifting of a long bar by two manipulators which do not have an IWV but are connected together through the bar that spans both the workspaces and permits cooperative action. The CWV therefore is dependent on the workpiece as well the Workspace Envelopes of the cooperating manipulators. The CWV may change with each workpiece being handled by the cooperating robots.

Stages in a Cooperating Task

Cooperating tasks are divided into three stages, the (1) Rendezvous, (2) Pre-Rendezvous and (3) Post-Rendezvous. These three stages of task cooperation takes place in the CWV of the participating robots. The rendezvous pose of an workpiece is any pose of the workpiece during which the cooperating manipulators interact simultaneously with the workpiece. Rendezvous trajectories are generated when the workpiece is manipulated in motion during the cooperative interaction. Rendezvous trajectories are therefore a succession of rendezvous poses of the workpiece.

The three stages of the cooperating task are shown in Figure 4.3. All three stages of the cooperating task are critical since they are executed in the CWV. The planned moves for the manipulators in the CWV should be executed in a guarded manner.

Rendezvous period of the task

During the rendezvous period of the task, the manipulated workpiece is controlled simultaneously by the cooperating manipulators. Depending on the type of cooperation, extensive kinematic and dynamic property updates are required between the cooperating manipulators for successful execution of the cooperating task.

Pre-rendezvous period of the task

The pre-rendezvous period of the task commences when the manipulated workpiece enters the CWV and terminates when the rendezvous period commences. During this period, the dominant manipulator controls the workpiece before rendezvous and the other participating manipulators starts the pursuit to arrive at the manipulated workpiece for rendezvous. Information exchanged between the manipulators consists of sensed kinematic properties (pose, velocity and acceleration) to coordinate the rendezvous.

Post-rendezvous period of the task

The post-rendezvous period of the task commences with the completion of the rendezvous period of the task and terminates with the manipulated workpiece leaving the CWV. One manipulator controls the workpiece during this period while the strategy for the other participating manipulator involves a non-interfering retraction from the workpiece trajectory. The sensed kinematic parameters have to be exchanged between the manipulators for successful post-rendezvous completion of the task.

<u>Dominance Factor</u>

In planning tasks for multi-manipulator cooperation, one of the manipulators should play the "lead" role in trajectory planning and control of the workpiece trajectories. Instead of tagging manipulators as "Master" and "Slave" we prefer using "dominant" and "non-dominant" manipulators in task planning. The dominance factor is a measure of the "dominance" of one manipulator over another. Different strategies are possible for determining dominance factors during the execution of a cooperating task. The user may explicitly specify the variation of the dominance factors during the execution of the task. For example, the dominance factor could be defined as a share of the manipulator's load relative to the full load of the workpiece,

$$\left.\begin{array}{c}\text{Dominance Factor}\\ \text{of a}\\ \text{Manipulator}\end{array}\right) := \frac{\text{Workpiece Load Shared by Manipulator}}{\text{Full Load of Workpiece}}$$

The dominance factors of the participating manipulators could be constantly changing throughout the duration of the task. During the rendezvous period for a baton-pass of a workpiece, for example, the load shared by the transferring manipulator would change from full load to zero while that of the receiving manipulator would change from zero to full load. The dominance factors of the transferring and receiving manipulators, therefore, change from 1 to 0 and 0 to 1 respectively. The manipulator with a higher dominance factor is tagged the dominant manipulator and used for position controlling the workpiece on its prescribed trajectory.

As mentioned earlier, the dominance factors help in determining the dominant manipulator in a cooperating task. Dominance factors also provide a measure of the various workpiece torques and forces to be shared between the cooperating manipulators. The dominant manipulator is always position controlled and is programmed to follow the computed sample points on a specified task trajectory. The non-dominant manipulator, on the other hand, will be position or force controlled to follow the task trajectory based on the actual, sensed poses of the dominant manipulator during task execution. The concept of dominant and non-dominant manipulators will be advantageous when dealing with systems of manipulators. The sum of the dominance factors in manipulator subgroups could be used to determine which group of participating manipulators is dominant in the multi-robot cooperating task.

4.7 Motion Planning Primitives for Single-Robots

The cooperative tasking portion of a planned task takes place in a narrow time window during the rendezvous period of task execution. A major portion of a cooperating task may be dedicated to a single robot handling one manipulated object, except during the rendezvous period when several robots handle the work object. The task planner developed for multi-robot workcells should, therefore, be able to specify single robot tasks as well. General motion planning primitives, consequently, must include motion planning for single robot operations.

For motion planning of single robot operations, we have identified three primitives--MOVETO, ATTACH and DETACH [Pal and Doty 1987]. These primitives are supported in all current robot languages in one form or another. Translating the primitives developed here into existing robot languages should be straight forward.

MOVETO

The MOVETO command moves the manipulator end-effector to the specified destination pose. All other parameters in the command are optional and will provide more specific moves if present. Most robot languages provide a move command and most of the variations possible with the optional parameters.

Syntax of the MOVETO command

```
MOVETO <destination> {from <start>}
                          {along <trajectory>}
                          {with <velocity_profile>}
```

<start> specifies the starting pose for the requested motion along a specified <trajectory> with predefined velocities of the end-effector in the <velocity_profile>.

ATTACH

ATTACH is used to either attach the workpiece to the end-effector of a manipulator or a tool to the end-effector or manipulator flange as the case may be. The ATTACH command is more general than the GRASP command generally available in most robot languages. When acquiring workpieces, the end-effectors are usually commanded to exert a specified amount of force to properly acquire the object. In the case of the ATTACH command, there is no provision for specifying the amount of force or any other associated control parameter. The additional specifications should come from a control primitive associated with the motion primitive. The ATTACH command assumes that the manipulator end-effector has been properly positioned for attaching the object by a previously executed MOVETO command.

DETACH

The DETACH command is used to relinquish control of a workpiece or tool. This command usually does not have any control parameters associated with it. The DETACH command also assumes that the workpiece or the tool being detached from the manipulator has been positioned in the proper pose before the DETACH command is issued.

4.8 Cooperating Robot Motion Primitives

The two cooperating robot motion primitives described in the taxonomy are static and kinetic manipulation. This classification is based on the type of interaction between the cooperating manipulators and the workpiece during the rendezvous period of the task. This section describes and models static and kinetic manipulation. Load sharing of the workpiece load between the cooperating manipulators, the control strategy used during cooperation and collision avoidance are the three primary concerns in cooperating robots. As mentioned earlier in the chapter, the collision avoidance problem is ignored in our planning. Before discussing static and kinetic manipulation, we will also discuss load sharing and control strategy aspects of robot cooperation.

Load Sharing in Cooperating Manipulators

Load sharing considerations are important in both, static and kinetic manipulations. During the rendezvous period when participating manipulators cooperate through the workpiece, the loads, forces and moments of manipulation should be shared between the manipulators. During cooperation, load sharing between the cooperating manipulators will exist however small the duration of the rendezvous period. Many heuristic methods of load sharing in closed kinematic chains have been studied [Orin and Oh 1981, Chand 1984, Chand and Doty 1984]. Chand [1984] shows that the load balancing problem in closed kinematic chains is over-specified and applies linear programming techniques to solve it. The load sharing criteria are situational and depend upon the workpieces and the payload capacities of the cooperating manipulators. Heuristic criteria, such as minimum energy expended, have been applied to multi-manipulator systems to aid load sharing computation. This work does not go into the details of resolving the issues of choosing and solving load-sharing heuristics. We assume that the load sharing criteria are a part of the cooperation specification and are either specified by the user or determined by the task planner. Load sharing is a part of the cooperation primitive specification and is normalized over all the N manipulators participating in the cooperation task. If lsp_i is the normalized load sharing specification for the ith cooperating manipulator, then

$$\sum_{i=1}^{N} lsp_i = 1 \qquad (1)$$

The normalization of the manipulator load sharing specification in equation (1) implies that the entire load of the workpiece will be supported by the cooperating manipulators throughout the duration of the task.

Compliance in Robot Cooperation

Cooperating robot tasks go through a cycle of specification, modelling and control. Various techniques can be applied in the control of cooperating manipulators. The control methods are never more critical than during the rendezvous period of the cooperating tasks. During the rendezvous period, the cooperating manipulators form a closed kinematic chain through the manipulated workpiece. There are multiple controllable entities in this closed kinematic chain and it leads to potentially unsafe situations if they are not properly controlled.

During cooperation tasks, the workpiece should be controlled either at a rendezvous pose or during a rendezvous trajectory. In order to achieve a rendezvous pose or trajectory, a dominant manipulator in the cooperating task must be position controlled to maintain the workpiece on the prescribed task trajectory. If all the participating manipulators were to be position controlled, control errors would introduce undesirable external forces on the manipulator end-effectors.

A non-dominant manipulator could be position or force controlled depending on the situation. In either case, a non-dominant manipulator will be controlled using measured workpiece poses and forces rather than the computed or specified values. This reduces the errors in the closed chain by accounting for the positioning errors of the dominant manipulator.

Undesirable external forces may be corrected in two ways.

1. If the errors and forces are small enough, they can be absorbed by the built-in structural compliances of the participating manipulators. This is called <u>passive compliant control</u>. Passive compliant control would permit the non-dominant manipulator to be position controlled. Position controlling the non-dominant manipulator makes the computations and control simpler and does not require force sensors and force control on the non-dominant manipulator.

2. In cases where the forces are too large to be ignored, or where the built-in structural compliances in the systems cannot absorb the forces, a force control algorithm should be applied to the non-dominant manipulator. This control is <u>active compliant control</u>. In certain instances of cooperation, regardless of the positioning errors, it may be desirable to enforce a force control scheme on the non-dominant manipulator to reduce the interactive forces between the cooperating manipulators and the workpieces.

Passive and active compliant control are control sub-classes of both static and kinetic manipulation tasking primitives. The specified task determines which degrees of freedom of the manipulator are to be position controlled or force controlled in the compliant control schemes. In passive compliance sub-classes, the dominant and non-dominant manipulators will be position controlled in all the degrees of freedom.

In the active compliance sub-class of the multi-robot primitives, certain degrees of freedom will be chosen for position control and certain others will be force controlled. Here, the compliance frame will be attached to the principal axes of the workpiece. When the workpiece is in motion, the dominant manipulator will be position controlled in all the degrees of freedom. The non-dominant manipulator may be position controlled in the direction tangential to the workpiece trajectory and force controlled in the normal and bi-normal directions.

Notations and Definitions

Multi-robot cooperating tasks will be defined using the workpiece as the reference for task specifications. The center of mass (cm) of the objects will be used to define the positions of the object and the principal axes attached to the center of mass define the reference axes for the workpiece and the task specifications. Figure 4.4 shows the definitions of some of the terms that will be used in our modeling of the multi-robot cooperation primitives,

i	: any cooperating manipulator in the task
d	: the dominant manipulators in the task
n	: the non-dominant manipulators in the task
N	: Number of manipulators participating in the cooperating task
W	: Weight of the workpiece being manipulated
F_i	: Force to be generated at the end-effector of the nip manipulator (6-vector of forces and moments)
F_t	: Force to be generated on the workpiece to maintain the desired tangential velocity profile of the task. Computed from the workpiece mass and inertias and the velocity profile (6-vector of forces and moments)
${}^{u}B_i$	: Description of Manipulator i base frame wrt Universal Frame
${}^{u}T_{cm}$	: Transformation for Workpiece Center of Mass frame wrt Universal Frame
${}^{cm}G_i$	: Grasp point i wrt Object Center of Mass
Unprimed quantities	: set or computed quantities
Primed quantities	: measured quantities

In our notation, the superscript specifies the reference frame relative to which the transformation for the subscripted frame is specified. In this paper, we will assume that the grasp transformations for the manipulators are determined by other techniques. The methods for choosing the grasp transformations and the regrasping of workpieces during the execution of cooperating trajectories is the subject matter of future research efforts.

Static Manipulation

In a static rendezvous, the workpiece must be at a stationary pose during the rendezvous period. During the pre-rendezvous and post-rendezvous periods of static manipulation, the workpiece is controlled only by the dominant manipulator and the planning for the non-dominant manipulator does not depend upon the workpiece. In a simple example later in this section, the tasks during the pre- and post-rendezvous periods are planned as independent single robot tasks except for the collision avoidance in the CWV's which is done by synchronization locks. The algorithmic modelling of static manipulation will therefore focus only on the rendezvous period of the task.

The syntax for specifying static manipulation is,

Static Manipulation Specification

```
STATIC_MANIPULATION of     <object>
                    by     <manipulator_1>
                       sharing <load_sharing_spec>
                       using   <dominance_factor_profile>
                                  :
                                  :
                    and    <manipulator_n>
                       sharing <load_sharing_spec>
                       using   <dominance_factor_profile>
                    at     <rendezvous_pose>
                    during <rendezvous_period>
```

Although the above static manipulation specification is based on 'n' manipulators, for the sake of simplicity, we will discuss the case of static manipulation between two manipulators.

Rendezvous period

Pose Computations:

Static manipulation tasks are defined by rendezvous poses of the workpiece. Assuming that known transformations define the manipulator base frame and the workpiece frame relative to the universal station frame, and choosing specific grasp transformations for the manipulators, the kinematic equation for the dominant manipulator at rendezvous is given by

$$ {}^{u}RP_{cm} * {}^{cm}G_{d} = {}^{u}B_{d} * {}^{d}T_{6} \qquad (2)$$

where ${}^{u}RP_{cm}$ is the rendezvous pose of the center of mass of the workpiece with respect to the Universal Frame. In equation (2), ${}^{d}T_{6}$ is the forward solution of the dominant manipulator. The forward solution of a manipulator describes the relative pose of the end-effector frame of a manipulator with respect to its base frame.

Solving the kinematic equation in (2), the unknown pose matrix for the dominant manipulator can be computed using

$$ {}^{d}T_{6} = {}^{u}B^{-1}{}_{d} * {}^{u}RP_{cm} * {}^{cm}G_{d} \qquad (3)$$

The reverse kinematic solution of equation (3) gives the pose of the end-effector for the dominant manipulator for achieving the rendezvous pose of the workpiece. This configuration of the dominant manipulator will be used to control the dominant manipulator to take the workpiece to the rendezvous pose. After taking it to the rendezvous pose, the actual configuration ${}^{d}T'_{6}$ of the dominant manipulator can be measured using the position sensors of the manipulator. This measured configuration of the dominant manipulator can be used to compute the measured pose of the workpiece using

$$ {}^{u}RP'_{cm} = {}^{cm}G^{-1}_{d} * {}^{u}B_{d} * {}^{d}T'_{6} \tag{4}$$

Using the measured rendezvous pose of the workpiece, the configuration for the non-dominant manipulator for cooperation can be determined using the following equation:

$$ {}^{n}T_{6} = {}^{u}B^{-1}_{n} * {}^{u}RP'_{cm} * {}^{cm}G_{n} \tag{5}$$

If the configurations of ${}^{d}T_{6}$ and ${}^{n}T_{6}$ are achievable without interference or collision for the participating manipulators the static manipulation task is feasible.

Force computations:

During static manipulation, the workpiece is stationary at the rendezvous pose during cooperation. Therefore, the only forces involved in the force computations are the six-vector of the workpiece load W. The inertial forces due to the motion of the workpiece does not have to taken into consideration. The load is distributed among all the cooperating manipulators, dominant and non-dominant using their load sharing specifications. Each manipulators' share of the total load is given by equation (6).

$$F_i = lsp_i * W \tag{6}$$

where F_i is the 6-vector of forces on the end-effector of the ith manipulator.

Signalling for static manipulation

During static rendezvous, the synchronization is effected by signalling between the manipulators. The controlling manipulator will achieve the rendezvous pose of the workpiece and signal the rendezvousing manipulator about the availability of the workpiece. The rendezvousing manipulator then arrives at the rendezvous pose of the workpiece, attaches the workpiece and signals the controlling manipulator about the acquisition of the workpiece. The controlling manipulator will relinquish the workpiece, retract and signal the rendezvousing manipulator about the retraction. After this last signal, both the manipulators will continue with their planned trajectories.

We describe synchronization using a simple example. Synchronization between two cooperating manipulators will be achieved with synchronization locks placed on the CWV. The locks will prevent more than one manipulator from moving into the CWV during the static manipulation. The progression of synchronization and control are indicated in the chart shown below. CM stands for the controlling manipulator and RM for the rendezvousing manipulator. This method is a very conservative way of achieving synchronization. Tasks are considerably slowed down as the manipulators wait for the locks to free up the CWV. Other, more efficient techniques are possible for achieving collision free static rendezvous.

Synchronization of Two Manipulators for a Static Exchange

Controlling Manipulator/s	Key	Rendezvousing Manipulator/s
Lock CWV Take workpiece to Rendezvous Pose Unlock CWV	CM	Monitor CWV lock and wait until CWV unlocked
Monitor CWV lock and wait until CWV unlocked	RM	Lock CWV Moveto Rendezvous pose Grasp Workpiece
	RENDEZVOUS PERIOD	
		Unlock CWV
Lock CWV Release Workpiece Retract out of CWV Unlock CWV	CM	Monitor CWV lock and wait until CWV unlocked
Continue with task	RM	Lock CWV Retract out of CWV Unlock CWV
	OPEN	Continue with task

Kinetic Manipulation

During kinetic manipulation, the workpiece will be in motion during the cooperative task. Kinetic cooperation is therefore described by a rendezvous trajectory during the rendezvous period of the task. The rendezvous trajectory specification includes a velocity profile for the trajectory

During the pre-rendezvous period of a kinetic transfer, the workpiece is controlled by the transferring manipulator. The receiving manipulator executes a pursuit algorithm to arrive at the workpiece at the beginning of the rendezvous period of the task. During the rendezvous period, the cooperating manipulators execute the rendezvous trajectory and share the load of the workpiece. During the post-rendezvous period, the receiving manipulator controls the workpiece and the transferring manipulator executes a guarded retraction trajectory.

The kinetic manipulation specification shown below provides load sharing and dominance factor profiles for each participating manipulator. The specification also includes the workpiece trajectory with its velocity profile and a window indicating the rendezvous period.

Kinetic Manipulation Specification

```
KINETIC_MANIPULATION of     <object>
                     by     <manipulator_1>
                       sharing <load_sharing_profile>
                       using   <dominance_factor_profile>
                                  :
                                  :
                     and    <manipulator_n>
                       sharing <load_sharing_profile>
                       using   <dominance_factor_profile>
                     on     <trajectory>
                       with <velocity_profile>
                     during <rendezvous_period>
```

During kinetic manipulation, tasks are specified by the time varying trajectories ${}^{u}WP_{cm}(t)$ traversed by the workpieces over the duration of the task from the beginning of the task at $t = 0$ to the end of the task at $t = T_{task}$. A particular segment of the task executed during the rendezvous period is called the rendezvous trajectory segment. The rendezvous trajectory segment begins at time $t = t_{rend\_begin}$ and ends at $t = t_{rend\_end}$.

$$^{u}RT_{cm}(t) = {}^{u}T_{cm}(t) \text{ for } t_{rend\_begin} <= t <= t_{rend\_end} \quad (7)$$

During the execution of the task, a tangential force $F_t(t)$, tangential to the specified trajectory is required to maintain the workpiece on the task trajectory. This tangential force can be computed at any instant during the task using the mass of the workpiece and the specified velocity profile for the task.

The time variation of the dominance factor, $df_i(t)$, of each cooperating manipulator is a part of the specification for kinetic manipulation. In general, when N manipulators participate in the kinetic manipulation, the dominance factors of all the manipulators participating in the cooperation task are normalized so that

$$\sum_{i=1}^{N} df_i(t) = 1 \quad (8)$$

Suppose there are two manipulators participating in the kinetic manipulation. The dominance factor of one of the manipulators has to be 1 before and after the rendezvous trajectory. By the above equation, the dominance factor of the other cooperating manipulator has to be 0 before and after the rendezvous trajectory.

Pre-Rendezvous period of the task

In the pre-rendezvous period, only the dominant manipulator will control the workpiece. The dominance factor of the dominant manipulator will be 1. Therefore, computations need not be performed for the non-dominant manipulator to hold the workpiece during the pre-rendezvous period. However, the computations for the non-dominant manipulator will be to track the workpiece measured pose as a knot point to arrive at the rendezvous. Techniques for tracking trajectories using moving knot points have already been researched [Taylor 1979].

Pose Computations

During the pre-rendezvous period, the workpiece executes the specified task trajectory given by

$$ {}^{u}WP_{cm}(t) \qquad \text{for } 0 <= t <= t_{rend\_begin} \tag{9} $$

For the dominant manipulator controlling the workpiece during the pre-rendezvous period, the kinematic equation can be written similar to the static manipulation case.

$$ {}^{u}WP_{cm}(t) * {}^{cm}G_{d} = {}^{u}B_{d} * {}^{d}T_{6} \tag{10} $$

Solving the unknown pose matrix for the dominant manipulator from equation (10),

$$ {}^{d}T_{6} = {}^{u}B^{-1}_{d} * {}^{u}WP_{cm}(t) * {}^{cm}G_{d} \qquad (11) $$

Equation (11) can be used to compute the end-effector pose of the dominant manipulator controlling the workpiece during the pre-rendezvous period. Using this end-effector pose, the reverse kinematic solution for the configuration of the dominant manipulator can be computed and position controlled using the computed configurations. The controlled positions of the manipulator will not necessarily coincide with the computed configurations. The actual measured configuration of the dominant manipulator (${}^{d}T'_{6}$) which were controlled to go to ${}^{d}T_{6}$ can be used to compute the sensed pose of the workpiece.

$$ {}^{u}WP'_{cm}(t) = {}^{cm}G^{-1}_{d} * {}^{u}B_{d} * {}^{d}T'_{6} \qquad (12) $$

This measured configuration of the workpiece will be used as intermediate knot points for computing the pursuit formulation for the non-dominant manipulator to arrive at the rendezvous.

Force Computations

During the pre-rendezvous period of the task, only the dominant controlling manipulator interacts with the workpiece and generates all the tangential forces required for maintaining the workpiece on the specified trajectory and velocity profile.

$$\left.\begin{array}{l}\text{End-Effector Forces to be}\\\text{generated and controlled}\\\text{by the dominant manipulator}\end{array}\right) = F_t(t) \qquad (13)$$

<u>Rendezvous period of the task</u>

During the rendezvous period of the task, both the manipulators participating in the cooperating task control the workpiece. It is critical during this period that the computations and control of the participating manipulators are performed accurately to achieve proper cooperation. During the rendezvous period, the pose computations for controlling the workpiece will drive the dominant manipulator controller while the measured pose of the workpiece will drive the non-dominant manipulator controllers.

Pose Computations

During the rendezvous period, the cooperating task is specified using, ${}^{u}RT_{cm}(t)$, the rendezvous trajectory of the workpiece specified in equation (7).

As mentioned earlier, the kinematic equation for the dominant manipulator during rendezvous can be specified using,

$${}^{d}T_6 = {}^{u}B^{-1}{}_{d} * {}^{u}RT_{cm}(t) * {}^{cm}G_d \qquad (15)$$

Equation (15) gives the kinematic solution for the dominant manipulator for realizing the workpiece pose on the specified task trajectory.

The dominant manipulator can be position controlled to achieve the desired workpiece pose. The measured workpiece pose ${}^{u}RT'_{cm}$ can then be used to compute the desired pose of the non-dominant manipulator for cooperating with the dominant manipulator.

$$ {}^{n}T_{6} = {}^{u}B^{-1}_{n} * {}^{u}RT'_{cm}(t) * {}^{cm}G_{n} \qquad (16)$$

Force Computations

The dominance factors provide a measure for load distribution between the cooperating manipulators. Since the summation of the dominance factors of the cooperating manipulators is normalized to be 1, the entire tangential forces required to maintain the workpiece trajectories can be met by the cooperating manipulators.

```
End-effector forces to be generated )
and controlled by the i'th          ) = F_t(t) * lsp_i(t) (17)
manipulator in the cooperating task )
{ dominant or non-dominant }        )
```

<u>Post-rendezvous period of the task</u>

During the post rendezvous period of the task the dominant manipulator is the only manipulator controlling the workpiece. The non-dominant manipulator need not even be involved in the cooperating task. Various retraction algorithms for the non-dominant manipulator can be devised and investigated. A basic requirement for such algorithms is to avoid collisions and not interfere with the rest of the task while cycling the non-dominant manipulator to repeat the motions on the next workpiece. In sum, motions planned for the non-dominant manipulator should be guarded in nature.

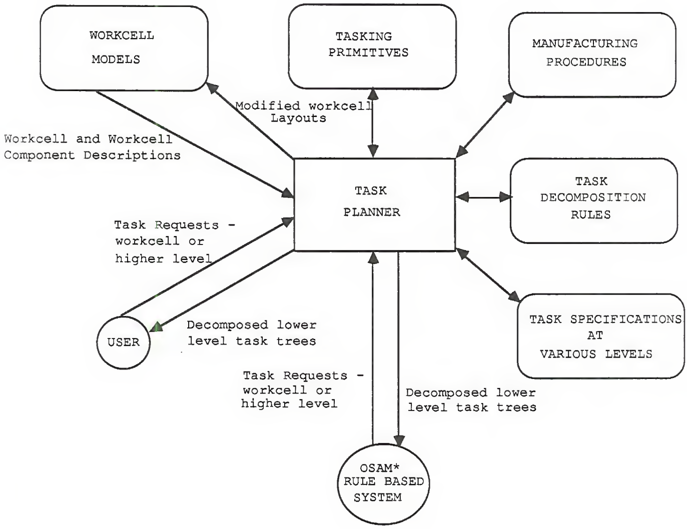

WORKCELL MODELS
TASKING PRIMITIVES
MANUFACTURING PROCEDURES
Modified workcell Layouts
Workcell and Workcell Component Descriptions
TASK PLANNER
TASK DECOMPOSITION RULES
Task Requests - workcell or higher level
USER
Decomposed lower level task trees
TASK SPECIFICATIONS AT VARIOUS LEVELS
Task Requests - workcell or higher level
Decomposed lower level task trees
OSAM* RULE BASED SYSTEM

FIGURE 4.1 Cooperating Multi-Robot Task Planner

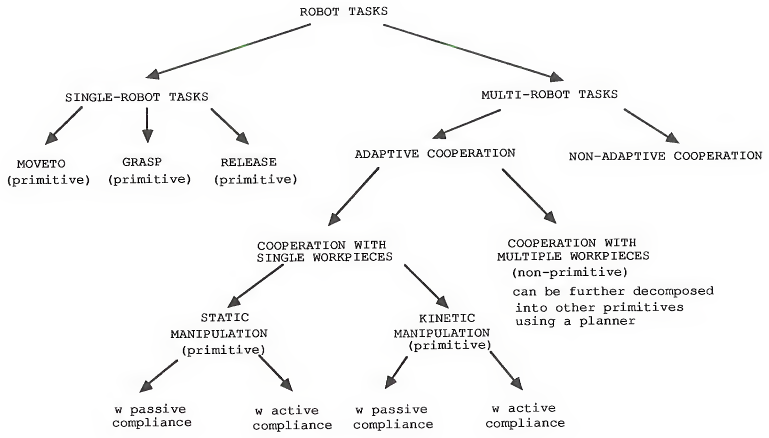


FIGURE 4.2 Taxonomy for Robot Motion Tasking Primitives

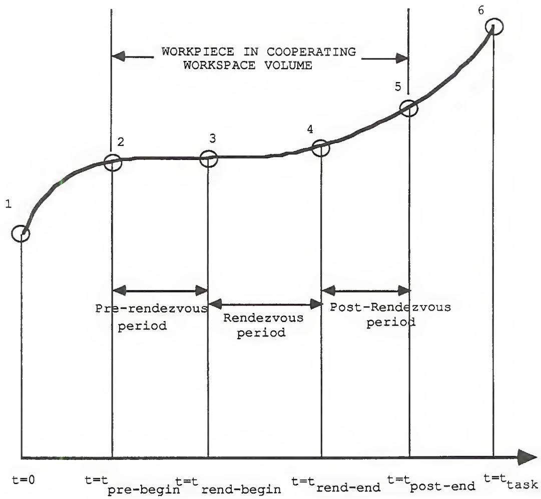


FIGURE 4.3 Stages in Cooperating Robot Task Execution

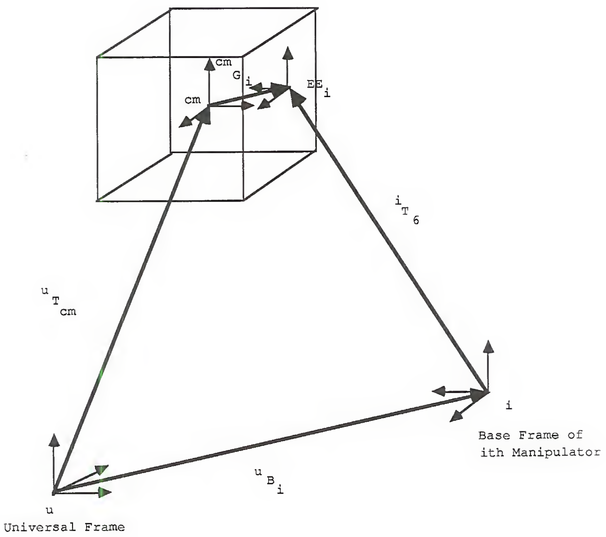


FIGURE 4.4 Definition of Frames for Cooperating Robot Primitives

CHAPTER FIVE

APPLICATION OF OBJECT AND TASK MODELLING BASIS

The primary motivation for this dissertation was to model and test cooperating robot tasks in multi-manipulator workcells. Due to the high risk nature of cooperating tasks in which the workspace environments of the manipulators are changing at a high rate, it is imperative that the tasks are simulated off-line before being committed to the workcells. In Chapters Three and Four of this dissertation, we developed tools for defining the world models and tasks in cooperating multi-robot workcells. In this chapter, we discuss an application and testing domain for the modelling and tasking primitives.

MEDL developed in Chapter Three provides a tool for describing the world models of manufacturing workcells. Simulation facilities which use workcell models require large volumes of data describing workcells and workcell components. We investigated a Knowledge Generation Schema for generating the more detailed information about workcell devices from a higher-level description of the devices. To describe object models to the knowledge generators, in essence, was the driving force behind the development of MEDL. Here, we describe the Robot Shell and Derived

Knowledge Generators for Manipulators as an application domain for MEDL.

In Chapter Four, we developed motion tasking primitives for cooperating multi-robot workcells. Numerical simulations were performed for verifying the feasibility of the motion tasking primitives for single-robot and two-robot workcells. The simulation facilities can be used in conjunction with a Task Planner to test multi-robot cooperation tasks based on the primitives developed in Chapter Four. The single- and two-robot simulation test-beds are also discussed in this chapter.

The Robot Shell, single-robot and two-robot Simulation Managers' users manuals are described in [Govindaraj 1987c]. Some of the user interaction screens from the Robot Shell and the Simulation managers are illustrated and discussed in Appendix 2.

5.1 An Application Domain for the Modelling Primitives

Manufacturing facilities and robot workcells are being pushed forward by Computer Integrated Manufacturing (CIM) technologies. Computerization of the manufacturing process is leading to more autonomous workcells. Workcell simulation systems require large databases of workcell and workcell component data [Mogal 1986, Hornick and Ravani 1986]. Database researchers have been addressing the issues involved in maintaining these large manufacturing databases [Su et al. 1988, Kemper and Wallrath 1987]. The database maintenance and application schemes assume that data for these large databases are available. Currently, such manufacturing databases are manually generated. As the CIM technology makes the workcell modelling autonomous, the need for automated workcell knowledge generation tools arises. One such tool, the Robot Shell, has been developed at the MIL [Govindaraj and Doty 1986]. The Robot Shell will become an integral part of the Manufacturing Workcell Modelling and Tasking Facility.

In this section, we describe the Robot Shell as an application domain for the Manufacturing Engineers' Description Language (MEDL) developed in Chapter Three of this dissertation.

Derived Knowledge Base

The system knowledge base refers to the body of information describing robot workcell components and their control. The knowledge base is used by the task simulation and control modules of the facility. The term "knowledge base" in the context of our usage does not yet imply the application of expert systems or artificial intelligence (AI) techniques in the development facility. We plan to employ such techniques in the future, however.

The need for robot system descriptions, or knowledge base, has been indicated by different researchers. Rembold uses a world model of a workcell in controlling robot systems using a general-purpose robot language (SRL) [Rembold and Blume 1984]. Kirschbrown uses a knowledge base for specifying tasks for robot systems and controlling task execution on these systems [Kirschbrown and Dorf 1984]. In spite of the potential applications for a knowledge base in robotics, most researchers either assume the availability of such a knowledge base or work with a limited subset of manipulators. One of our goals is to provide a development facility that allows users to simulate, integrate and test any collection of manipulator and workcell components available in the marketplace. This goal excludes using a limited knowledge base for a specific vendor.

The large amount of information to be stored in the robot systems knowledge base as we envision it makes manual entry a tedious task. The robot system development facility automates the knowledge base generation by deducing low-level descriptions and mathematical models of robot systems components from their high-level specifications. The automatic modeling reduces the chances of error in the tedious process of algebraic manipulation, particularly symbolic inversion and multiplication of homogeneous matrix sequences [Malm 1984].

Two schemes model the generation and application of the Derived Knowledge Base for robot system components. The Knowledge Base Generation scheme (Figure 5.1) is common for all the Manufacturing Workcell modelling basis and aggregations described using MEDL.

A user enters data about an object into the Object Specification Catalog by means of interactive menus and screens. These data, along with the object model, guide the Knowledge Generator when it creates the Derived Knowledge Base. The object model defines what object data the user must enter so that the Knowledge Generator can create the Derived Knowledge Base for that object. An important consideration here is to insure that object data entry is convenient and the requested data meaningful to the user. The Manufacturing Engineer's Description Language (MEDL) and the MEDL interpreter (MEDLI) will provide a good vehicle for describing the object models as described in Chapter Three.

We illustrate this concept of knowledge base generation with an example. The object model for a manipulator, an instance of an actuated kinema, dictates the specification of the geometric shape and dimensions along with the kinematic and dynamic properties of the manipulator's mechanical structure. After all the data required by the model is entered by the user into the Object Specification Catalog, the knowledge generator can derive a detailed description of the object.

The kinema object model controls the user interface for the generation of all the information for an object instance in the specification catalog. The kinema instances are defined in the Derived Knowledge Base by A-matrices, forward and reverse solutions, the inertia matrix and the dynamics equations. The kinema model specifies these Derived

Knowledge Base requirements and controls the Kinema Knowledge Generator to generate this information from the user specifications of the kinema instance.

Figure 5.2 shows our Derived Knowledge Base application scheme. Application programs in the Robot System Development Facility can only use the Derived Knowledge Base and cannot modify it. Such programs can display the symbolic knowledge base information to the user or perform numerical evaluations of the symbolic equations. For example, a task execution program might request the knowledge base to provide a numerical, reverse kinematic solution to a particular manipulator and also the appropriate control signals required by the manipulator's controller to generate the computed joint motion.

The symbolic portion of the knowledge base is stored in ASCII text code. A symbolic textual knowledge representation scheme was chosen to allow robot system knowledge bases to be portable. This permits different operating systems supporting the development facility and the different programming languages that code the control and simulation routines to access the knowledge base. Another advantage of an ASCII symbolic knowledge base is the availability of standard text handling programs for updating and maintaining the knowledge base. The user can also view the data base and make sense of it.

Symbol Manipulation

The role of digital computers as "number crunchers" has been firmly established. Nonetheless, computer programs are equally adept at manipulating algebraic expressions in which abstract symbols represent numerical values. Carrying out the computations symbolically instead of making numerical substitutions during intermediate stages of the computation provides several advantages [Pavelle et al. 1981].

(1) Although numerical values may produce the required end result, symbolic intermediate results may provide better insight into the computational process.

(2) Whether simplifications are performed manually or automatically, evaluation of the numerical values of algebraically simplified expressions results in economical use of computer time.

(3) Roundoff errors introduced by numerical approximations accumulate and produce results which may not be as exact as algebraic results.

Programs of varying degrees of complexity, ranging from MUMATH for personal computers to MACSYMA for special-purpose LISP machines, have been written for performing symbolic computations. Available symbolic programs manipulate expressions satisfying the rules of algebra and calculus.

Dedicated symbol manipulation programs usually run as stand-alone applications and execute slowly in multi-user or multi-tasking environments. MACSYMA [Bogen 1977], for example, allows symbolic integration and differentiation, linear or polynomial equation-solving, algebraic simplification and factoring, solution of differential equations, tensor and matrix multiplication, and a host of other symbolic mathematical operations. The complexity of the general purpose symbol manipulation programs usually make them large.

Robot manipulator kinematics and dynamics equations depend heavily on homogeneous matrix and vector manipulation. Manual development of symbolic versions of these equations for each robot is tedious and error prone. On the other hand, general numerical solutions may not be as efficient as solutions based upon algebraically simplified equations of motion for a specific robot. Available symbol manipulators do not appear to be the solution to these problems. Commercially available symbolic manipulators appear to provide more processing power than required and do not permit easy interfacing to our programs. Our approach, consequently, has been to develop our own simple, symbolic-manipulation software package to generate symbolic expressions that model robot workcell components. The collection of the symbolic expressions makes up the robot system symbolic knowledge base. These symbolic operation procedures are called by knowledge base generation programs to derive programs and infer lower level descriptions based upon object models.

Our symbol manipulator handles 4 X 4 homogeneous matrices, vectors and scalars. The more common dyadic operations required by the knowledge base generation programs involving matrices, vectors and scalars are supported. Monodic operations for performing the transpose and computing the inverse of homogeneous matrices are essential for the robot system knowledge base generation. Functions supporting these operations have also been developed. A command interpreter provides a stand-alone symbol manipulator to conduct research and test functions under development.

Robot Shell

The phenomenal growth in UNIX installations attests to its popularity. The large number of software development tools available in UNIX along with the portability of the programs makes it an ideal operating system for developing large programs. UNIX's flexible and reconfigurable command interpreter called a "shell" provides a powerful user interface to the programs available in the operating system. The UNIX operating system has been used before to provide a standard interface to a set of tools for controlling robots [Hayward and Paul 1986].

The Robot Shell provides the user interface to our development facility. UNIX makes user login parameters available to shell procedures that enables the robot shell to tailor the development facility prompts and responses according to the user. The robot shell provides two modes of interaction for the user. A purely interactive mode which is menu-driven and a command-interpreter mode that allows the user more liberties and flexibility. As more functions and modules are added to the development facility, the shell can be easily modified to accommodate them.

Robot Workcell Description Module

Reprogramming robots to changing manufacturing procedures is most often achieved by keeping the workcell unchanged and modifying the robot control programs to perform the new tasks. Due to the high monetary and time expenses involved, the workcell configuration is seldom modified. The robot system development facility reduces the time cost of workcell changes by providing off-line design and verification tools. The verification process does not interrupt the on-going manufacturing process. Since task descriptions depend on the workcell configurations, workcell descriptions are required for robot task verifications as well. Robot workcells and workcell components can be described using MEDL. A Kinema Knowledge Generator has been implemented and tested to verify the concept of knowledge generation and the application of object models described using MEDL. The kinema knowledge generator will be described in this section. The knowledge generators can be extended for other classes of robot system components as well. The MEDLI implementation will use the kinema knowledge generators as an experimental test-bed and extend them for general, user-specified MEDL object models.

The Kinema Knowledge Generator

Homogeneous matrices conveniently describe kinematic chains [Paul 1981]. The Kinema Knowledge Generator uses homogeneous matrices, called A-matrices to define its kinematic structure.

Kinema control is broken down into kinematic and dynamic control. Kinematic control is used to move the manipulated objects in the workcell at specified velocities and accelerations without considering the torques and forces in the joints required to make the moves. Dynamic control includes consideration of torques and forces in the motion. Figure 5.3 shows the sub-modules of the kinema knowledge generator. Manipulators are described by their kinematic structure, link and mass properties and then cataloged. These catalogs maintain symbolic or numerical information.

Table 1 displays a simplified object model for an instance of the Kinema Knowledge Generator. The example illustrates for a specific kinema instance, a manipulator, the information contained in the different submodules of the

Kinema Knowledge Generator. For other kinema instances, some of the submodules may be redundant or have null entries in the kinema Derived Knowledge Base.

Table 1   Kinema Knowledge Generator

```
Component Instance: Manipulator

Specification Catalog:
   Object     : 1. Geometric Specifications
                2. Kinematic Properties
                3. Dynamic Properties
   Controller : 1. Inputs
                2. Outputs

Derived Knowledge Base Information:
   A-Matrices, Forward Solution (Tn), Reverse Solution,
   Inertia Matrix, Dynamics Equations
```

Currently available kinematic sub-modules are the A-Matrix Generator, the Forward Solution, the Reverse Solution and the Jacobians Modules. The A-Matrix Generator uses the structural description of the manipulator found in a kinema specification catalog to create the A-matrices for the links of the manipulator. These A-matrices from the knowledge base are also used by other sub-modules of the Manipulator Description Module.

The Forward Solution Module multiplies the A-matrices of the successive links of the manipulators to form the forward solution matrix (Tn) of the manipulator [Paul 1981]. Tn converts the joint configuration of the manipulator into the position and orientation of the end-effector in the manipulator frame. The Reverse Solution Module computes the joint configuration of the manipulator when the desired end-effector coordinates in the manipulator frame are specified. Determining the closed-form symbolic reverse solution depends upon the manipulator, is quite complex and, at present, lacks analytic generality.

The current version of the Reverse Solution module identifies the candidate terms from the matrix equations needed to solve for the successive joint angles of the manipulator. Even this user assistance substantially reduces the effort to search for terms that will yield the reverse solution.

The Jacobian module performs the symbolic computation of the end-frame and the user-specified intermediate frame Jacobians of the manipulator. The Jacobian assists in incremental analysis and motion of the manipulator. The inverse Jacobian identifies the singularity configurations of the manipulator in its work-envelope and hence determines the manipulator configurations where velocities may be constrained.

5.2 A Testing Facility for the Motion Tasking Primitives

In Chapter Four of this dissertation, we described the motion tasking primitives. Robot motion commands were generally broken down into single-robot and multi-robot motion tasking commands. In multi-robot motion commands, we identified static and kinetic manipulation as the two motion primitives. Numerical simulations were used for testing the feasibility of the above motion primitives. A simulation test-bed has been developed for testing single- and multi-robot motion primitives. The simulation test-bed has been integrated as a part of the Manufacturing Workcell Modelling and Tasking Facility being developed by the MIL. A Task Planner for cooperating multi-robot workcells is also under development at the MIL [Pal and Doty 1987]. The Tasking Primitives Testing Facility will also be used as a test-bed for the Task Planner.

A Simulation Facility for Single-Robot Workcells

A highly-interactive simulation manager for single-robot workcells has been developed as a part of the Tasking Primitives Testing Facility. The simulation manager permits numerical forward and reverse solutions and kinematic and dynamic path planning for single robots. The users are allowed the flexibility of controlling the simulation environment and results to suit their tastes and requirements. For example, data entry and display of points in the workspace could be done in the cartesian or configuration space of the robots. Configuration angles of the rotary degrees of freedom of manipulators can be displayed in degrees or radians. Any screen in the facility can be printed to generate on-demand hardcopies of the simulation results. The configuration sample points during trajectory planning could either be plotted on the AT&T 5620 dot mapped display (DMD) screen or printed to generate hardcopies.

The generation of numerical forward and reverse solutions is pretty straight forward. In the case of the reverse solution, however, there is no unique reverse solution for a robot manipulator. The reverse solutions are usually in the form of a binary tree. Researchers suspect that 16 is an upper bound on the number of solution sets for a general six degrees-of-freedom manipulator. Research, currently in progress at the MIL, has proven that some manipulators do have 16 solutions [Manseur and Doty 1987].

In the simulation manager, on numerical reverse solutions, the entire solution tree is displayed showing the user all the possible configurations for achieving a specified cartesian pose for the end-effector of the manipulator. The simulation manager also permits sets from the solution tree to be picked to evaluate their end-effector poses. This helps to verify the validity of the reverse solution computations.

In the trajectory generation portions of the simulation manager, it is possible to apply Richard Paul's [Paul 1979] or Taylor's [Taylor 1979] schemes for computing the end-effector poses during intermediate sample points on the trajectory. The user may specify any sampling rate for the simulations and will receive intermediate configuration updates at that rate. In both schemes, a fly-by computation is performed for the trajectory corner points without going through the specified corner poses. Since the intermediate sample point computations for the trajectories are performed in cartesian space, a reverse solution has to be computed at every sample point to determine the configurations of the manipulator. One solution has to be selected for the manipulator from the reverse solution tree. Various methods may be applied in picking the appropriate solutions. In our simulations, we use a weighted solution picking strategy which is described in [Govindaraj 1987c].

Dynamics computations can also be performed for both trajectory planning schemes. In dynamics simulations, the joint torques required to maintain the end-effector on the specified trajectory are computed. The simulation manager provides a choice of Lagrangian or Newton-Euler methods of dynamics computations.

The single-robot simulation manager uses the Unimation Puma-260 manipulator as the simulation robot. The robot dependencies are localized to a sub-module of code. Therefore, to simulate another robot using the simulation manager, it is as easy as writing a forward and reverse solution procedure for the new robot and defining the inertial parameters of the new robot.

The simulations result in configurations for the Puma-260 in the joint space. This six-vector of joint configurations can be used to drive 3D graphical images on a graphics workstation. The same configuration results can also be used to drive real Puma-260 robots through a reverse-engineered VAL controller. The Puma-260 simulation manager has been implemented with 4800 lines of C language code and is described in a user's manual [Govindaraj 1987c].

A Simulation Facility for Two-Robot Workcells

A two-robot simulation test-bed has been built based on the single-robot simulation test-bed described in the previous section. Two Puma-260 manipulators are placed in the simulation facility. Like the single-robot simulation manager, the two robot simulation robot dependencies are also localized in one sub-module. Therefore, simulations can be performed for any robot manipulator. In the two-robot simulation manager, the trajectories are specified relative to a Universal Frame (u) instead of a robot base frame. Any of the parameters uB1, uB2, cmG1 and cmG2 defined in Chapter Four can be changed. This provides the simulation manager the flexibility to tailor the positioning of the manipulators in two robot workcells and the choice of arbitrary workpieces in the tasks.

The two robot simulation manager permits the testing of static and kinetic manipulation. Computations for static manipulation are performed for both the Puma-260's. The computed configurations for both the manipulators for the rendezvous pose of the workpiece are displayed. For kinetic manipulation, Paul's trajectory computation scheme is applied for the workpiece trajectories. The simulation outputs of the kinetic manipulation shows the workpiece entering the cooperating workspace volume, and shows the configurations of the two manipulators during the pre-rendezvous, rendezvous and post-rendezvous portions of the kinetic manipulation.

The two robot simulation manager is also written in the C language. The current version of the simulation test-bed consists of 1000 lines of code in addition to the common code used from the single robot simulation manager. The procedures for static and kinetic manipulation simulations are written with parameter passing as described for their specifications in Chapter Four. This makes it possible for the Task Planner of the Manufacturing Workcell Modelling and Tasking Facility to be able to use this test-bed for verifying the task trees generated for planning multi-robot cooperating tasks.

5.3 Robot Workcell Modelling and Tasking Facility

The application domains for the modelling and Tasking primitives were described in the previous sections of this chapter. Apart from providing a test vehicle, these useful tools have been combined with other programs in the Robot Shell to form the initial implementation of the Manufacturing Workcell Modelling and Tasking Facility. The Facility uses the Robot Shell as the user interface and provides simple modelling, and simulation capabilities to the user. The current implementation of the facility has been useful for establishing the feasibility of building such a facility and also provides a useful tool for the researchers in the MIL. In this section, we describe the main modules in the current implementation of the Manufacturing Workcell Modelling and Tasking Facility and the current implementation status.

Robot Workcell Modelling and Tasking Facility Description

The Robot Workcell Modelling and Tasking Facility is a collection of software tools designed to provide users with a simple and flexible method for designing, integrating and testing robot workcells and tasks. Users may define any robot system component they wish to be modelled. This generality includes modelling multi-manipulator workcells and cooperating tasks within such workcells.

The novice user is gently led with user friendly messages, prompts and choices that mask the lower-level detail. As the user becomes more knowledgeable, he is treated intelligently with a professional friendly interface to the software system. A professional friendly-interface gives users access to all the information and tools in the facility and does not overburden them with simple messages and prompts.

The knowledge base generated by the workcell description module is used by the other software tools of the robot development facility. The user interface recognizes the requested action and identifies all the information required for processing the request. It may not be "smart" enough to know how to perform all the user requested tasks, but it is smart enough to recognize the information needed to perform a command and queries the user for additional information. User intervention is requested for any task for which the data in the knowledge base is incomplete and cannot be generated using the available information.

The major software components of the robot system and task development facility (Figure 5.4) are the

1. Workcell Description Module
2. Task Description Module
3. Robot Simulation Module
4. CIM Network Interface Module
5. Code Generator Module

Implementation Status of the Facility

The Robot Workcell Modelling and Tasking Facility is the result of 5 man-years of effort at the Machine Intelligence Laboratory. It began as a collection of software tools required to ease the task of robotics research in multiple kinematic linkages. All the kinematics aids are currently supported and the dynamics aids are under development. The Code Generator supports the completed kinematics modules. The low-level simulation routines for forward and reverse solutions with general interfaces into the compiled modules and powerful user interactions has been implemented.

Standard network interfaces are being constructed and studied using this robot control network as a basis. Without counting the various software development tools provided by UNIX and used by the Robot Shell, the "home grown" shell currently consists of about 10,000 lines of code in the PASCAL and C languages.

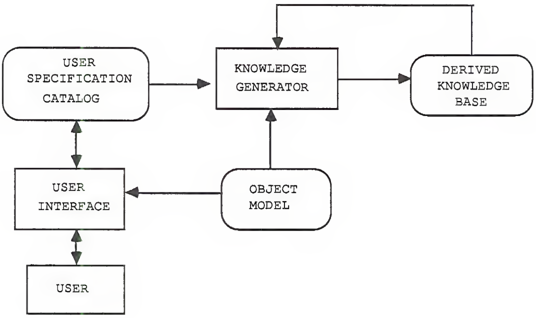


FIGURE 5.1 Derived Knowledge Base Generation Scheme

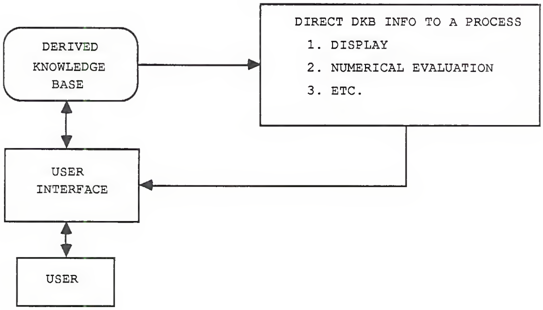


FIGURE 5.2 Derived Knowledge Base Application Scheme

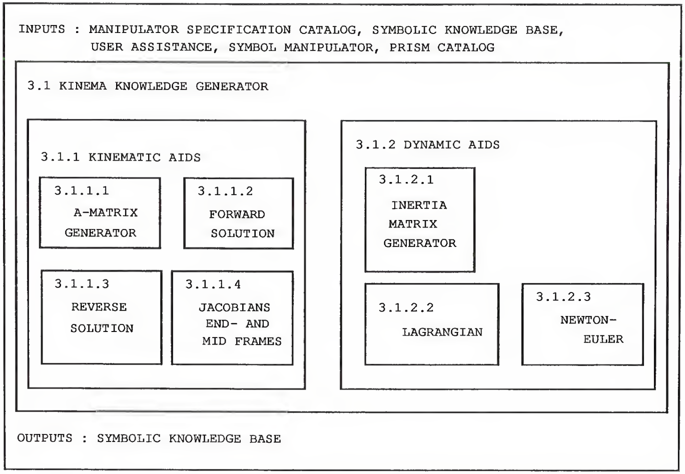

INPUTS : MANIPULATOR SPECIFICATION CATALOG, SYMBOLIC KNOWLEDGE BASE,
USER ASSISTANCE, SYMBOL MANIPULATOR, PRISM CATALOG
3.1 KINEMA KNOWLEDGE GENERATOR
3.1.1 KINEMATIC AIDS
3.1.1.1 A-MATRIX GENERATOR
3.1.1.2 FORWARD SOLUTION
3.1.1.3 REVERSE SOLUTION
3.1.1.4 JACOBIANS END- AND MID FRAMES
3.1.2 DYNAMIC AIDS
3.1.2.1 INERTIA MATRIX GENERATOR
3.1.2.2 LAGRANGIAN
3.1.2.3 NEWTON-EULER
OUTPUTS : SYMBOLIC KNOWLEDGE BASE

FIGURE 5.3 Kinema Knowledge Generator

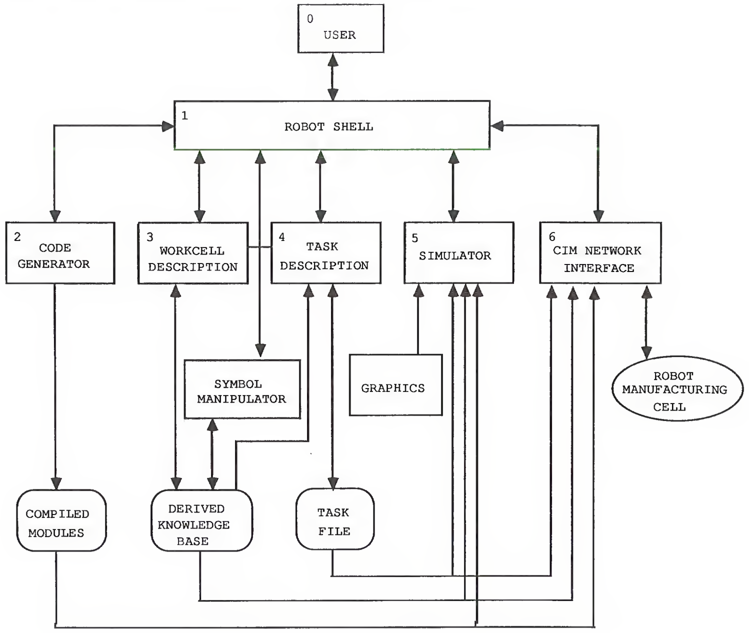


FIGURE 5.4 Robot Workcell Modelling and Tasking Facility

CHAPTER SIX

CONCLUSION

6.1 Summary

In this dissertation, we studied the modelling of cooperating multi-manipulator workcells. The modelling task was split into two parts, object and task modelling. For both, object and task modelling, we developed language and system independent primitives. This independent basis will provide tools which are absolutely essential to integrate the different modelling tools available to a manufacturing engineer.

Although the different phases of manufacturing product development life-cycle are adequately supported by software tools, these tools have not been integrated into one viable facility for a manufacturing engineer. The lack of integration is basically due to the various languages and interfaces supported by the different tools. Manufacturing Automation Protocol (MAP) is a step in the right direction towards integrating various automation tools. MAP provides the software protocol and hardware specifications for different automation products to communicate data between each other. Although this gives data portability, users still have to learn and use the various systems in the different system languages.

In Chapter Two, we developed a total integration product which supports automation from design to simulation and finally to execution in actual manufacturing facilities. In that design, we established the requirements for workcell modelling and workcell tasking tools which were developed next in the dissertation.

In Chapter Three, we presented a world modelling basis for multi-manipulator workcells. We developed a Manufacturing Engineers' Description Language (MEDL) which was system and implementation language independent. We identified five basic elements--statica, informata, sensors, actuators and processes--which form the representational core of our modelling language. An object modelling methodology based on aggregations and aggregation rules was also described with examples.

There are many robot manufacturers and just as many robot languages in the industry. None of the currently available robot languages supports multi-robot cooperation tasks. Instead of developing a new robot language that supports multi-robot cooperation, we designed a new methodology for language independent task definition for robots. We developed robot independent motion tasking primitives to support multi-robot cooperation tasks. A taxonomy for motion tasking primitives was defined and the multi-robot tasking primitives at the command level were designed and simulated. The multi-robot motion tasking primitives can be used along with perceptor and control tasking primitives in a Task Planner being developed at the MIL.

As part of the Manufacturing Workcell Modelling and Tasking Facility described earlier, large databases of manufacturing device data are required. We describe a methodology for the automated generation of lower-level device data from high-level specifications of the devices. We call this the derived knowledge generation scheme. This knowledge generation scheme was tested using manipulators as an example.

Provisions are made in the knowledge generation scheme to define new classes of objects using MEDL object models. A MEDL Interpreter is currently being developed at the MIL. This MEDL interpreter will be used to provide object models for the knowledge generators of other devices.

We also have developed a simulation test-bed for testing motion tasking primitives for multi-robot cooperation tasks. This same simulation test-bed will also provide a test-bed for a Task Planner under development at the MIL. The knowledge generator, a symbol manipulator program and the simulation test-bed software modules are part of a first-level implementation of the Manufacturing Workcell Modelling and Tasking Facility.

6.2 Recommendations for Future Work

The design of the Manufacturing Workcell Modelling and Tasking Facility has provided many insights into areas of further research. We have set up a modelling sub-group at the MIL for further continuing research efforts in this area. The results of this dissertation has helped direct the efforts of this MIL sub-group. Two of the primary areas of research are the Modelling facility and the Tasking facility. In this section, we discuss future work in both these areas and in robot cooperation.

Modelling Basis

1. A workcell modelling language (MEDL) was described in this dissertation. Further work in the MIL is focusing on developing a MEDL Interpreter (MEDLI) which will convert MEDL descriptions to OSAM* s-diagrams. MEDLI will be developed assuming OSAM* as an underlying object-oriented data management schema for the Manufacturing Workcell Modelling and Tasking Facility.

2. MEDLI can also be written to generate object models for Knowledge Generators.

3. Knowledge generators for other types of objects, both basis and aggregates, have to be written to develop automated derived knowledge generation for a whole class of MEDL described devices.

4. MEDL is intended for building object models and is inherently static in nature. Extensions to MEDL can provide dynamic modelling capability for monitoring object models changes.

Tasking Primitives

1. In this dissertation, we only developed the motion tasking primitives for multi-robot cooperation. Future work has to be done on perceptor and control primitives.

2. A general purpose task planner for planning multi-robot cooperation tasks using motion, perceptor and control tasking primitives has to developed. MIL is currently investigating a task planner for just motion task requests in multi-robot workcells. The task planner will analyze tasks down to the motion primitives, test the tasks in the two-robot simulation test-bed described in Chapter Five and transfer it for testing in a two-robot minimover-5 workcell.

Robot Cooperation

In this dissertation, we approached robot cooperation from the command and tasking level. As mentioned earlier, other research efforts are directed towards the control level of robot cooperation. Results of that body of research is applicable to and will be assimilated by the motion tasking primitives. For future work at the MIL, we only suggest a few areas which have been overlooked at the lower level of cooperation.

1. The first area is in grasp point analysis. In our tasking primitives, we assumed that the grasp point on the workpiece was known as a transformation specified by the task. We did not investigate the strategies in choosing grasp transformations. The selection of grasp transformations for multi-robot operation may be a challenging future task.

2. Apart from grasp transformations, changing grasp during task or re-grasping is entirely impossible during the rendezvous periods of a task. This will affect the planning of the rendezvous periods of cooperating tasks. Future research is recommended to investigate the actions to be taken when re-grasping is unavoidable during rendezvous of a task.

3. Related to the grasp point analysis problem is solution picking strategies for multi-robot cooperation. As we mentioned in Chapter Five, picking solutions out of the multiple solutions of a reverse solution tree is a difficult problem even with single robots. Various heuristics can be devised and applied for solution picking strategies. This problem becomes more complex when solutions have to be picked for multiple cooperating robots. Future research is suggested in investigating heuristics for picking solutions for cooperating multiple robots.

GLOSSARY

Abstract: Logical properties of objects. Can be modelled as information.

Actuators: MEDL functional basis used to model abstract to concrete transducer objects.

Aggregation: MEDL mechanisms for defining classes of devices in manufacturing workcells.

Basis: Fundamental entities in the object modelling language. Used as building blocks for object descriptions.

Concrete: Physical properties of objects. Can be modelled as rigid bodies.

Cooperating Workspace Volume: A workpiece structure dependent volume where the cooperating manipulators can interact with the workpiece at the requested grasp transformations.

Cooperation: More than one manipulator working towards a common goal.

Derived Knowledge: Lower-level instantiation information derived from higher-level user specifications.

Dominant manipulator: Position controlled cooperating manipulator.

Informata: MEDL functional basis used to model logical properties of objects.

Instances: Specific members of the classes of devices modelled using MEDL.

Kinetic manipulation: Cooperative manipulation of workpieces in motion.

MEDL: Manufacturing Engineers' Description Language. A high-level, object modelling language.

Non-Dominant manipulator: Position or force controlled cooperating manipulator.

Object: Manufacturing workcell devices and collections of devices.

Processes: MEDL functional basis used to model abstract action parts of objects.

Rendezvous: Period of cooperation task during which all cooperating manipulators interact simultaneously with the workpiece.

Sensors: MEDL functional basis used to model concrete to abstract transducer objects.

Static manipulation: Cooperative manipulation of stationary workpieces.

Statica: MEDL functional basis used to model rigid body properties.

Task: Commanded action in a manufacturing workcell. Tasks can be specified at various levels of detail.

Task Primitives: Planner recognized actions which are manipulator language independent. Used as building blocks for constructing tasks at a higher level of abstraction.

APPENDIX 1

DESCRIPTION OF MEDL

MEDL Program Structure

```
DESCRIPTIONS:

list_of_aggregations

DECLARATIONS:

list_of_instantiations

list_of_aggregations ::=   aggregation
                  ¦ list_of_aggregations aggregation

list_of_instantiations ::=   instantiation
                  ¦ list_of_instantiations instantiation
```

MEDL Declarative Part

```
instantiation ::= instance_list ':' instance_type ';'

instance_list ::=   instance_name
                  ¦ instance_list ',' instance_name

instance_name ::= identifier

identifier ::=   letter
               ¦ identifier letter_or_digit

letter ::= lower_case_letter ¦ upper_case_letter

digit ::= '0' ¦ '1' ¦ ...... '9'

underscore ::= '_'

letter_or_digit ::= letter ¦ digit ¦ underscore
```

```
instance_type ::=   basis
                  | aggregate_name

aggregate_name ::= identifier
```

MEDL Basis Definitions

```
concrete_properties ::= geometry   color     texture
                        material   WEIGHT    DENSITY

geometry ::= shape   VOLUME   AREA   INERTIAS
                     BODY_ATTACHED_FRAME

shape ::= data_structures defining shapes
              (implementation CAD system dependent)

color ::=  data_structures defining color
              (implementation CAD system dependent)

texture ::= data_structures defining texture
              (implementation CAD system dependent)

material ::= data_structures defining material
              (implementation CAD system dependent)

abstract_properties ::= data_structures
              (object-oriented data structures or
               data structures based on the
               implementation programming language)

basis ::=   statica
          | informata
          | sensor
          | actuator
          | process

statica ::= concrete_properties

informata ::= abstract_properties

sensor ::= concrete_sensor   abstract_sensor

concrete_sensor ::= concrete_properties

abstract_sensor ::= abstract_properties
```

```
actuator ::= concrete_actuator   abstract_actuator

concrete_actuator ::= actuator_description  actuator_action

actuator_description ::= concrete_properties

actuator_action ::= LINEAR_MOTION ¦ ANGULAR_MOTION

abstract_actuator ::= abstract_properties

process ::= process_input  process_plan  process_output

process_input ::= abstract_properties

process_output ::= abstract_properties

process_plan ::=   servo_level_plan
                 ¦ command_level_plan
                 ¦ task_level_plan
                 ¦ planning_level_plan
                 ¦ goal_level_plan

concrete_basis ::=   statica
                   ¦ concrete_sensor
                   ¦ actuator_description

abstract_basis ::=   informata
                   ¦ abstract_sensor
                   ¦ abstract_actuator
                   ¦ process
```

MEDL Aggregation Descriptions

```
aggregation ::= aggregate_name 'DESCRIBED_BY'
                                    aggregation_rule ';'

aggregate_name ::= identifier

aggregation_rule ::=   concrete_declaration  '&'   <NULL>
            ¦ <NULL>                 '&' abstract_declaration
            ¦ concrete_declaration '&' abstract_declaration

concrete_declaration ::=
     (CONCRETE_PART)
           concrete_name DEFINED_BY concrete_basis
                      ( AND concrete_object )

concrete_name ::= identifier

concrete_object ::= concrete_connections
                  ¦ concrete_object  concrete_connections
```

```
concrete_connections ::= AND <COMBINE> concrete_name
                                AT [homogeneous matrix]
                   | AND <CONNECT> concrete_name
                                AT [homogeneous matrix]
                   | AND <DRIVE> <CONNECT> concrete_name
                                AT [homogeneous matrix]
                                USING actuator_action
                   | AND <COMBINE> concrete_basis
                                AT [homogeneous matrix]
                   | AND <CONNECT> concrete_basis
                                AT [homogeneous matrix]
                   | AND <DRIVE> <CONNECT> concrete_basis
                                AT [homogeneous matrix]
                                USING actuator_action

abstract_declaration ::= abstract_name DEFINED_BY
                                       abstract_object

abstract_name ::= identifier

abstract_object ::=   abstract_basis
                    | transformation
                    | abstract_name
                    | abstract_object  AND  abstract_basis
                    | abstract_object  AND  transformation
                    | abstract_object  AND  abstract_name

transformation ::= <TRANSFORM> src_of_abstract_inputs  TO
                               dest_of_abstract_outputs
                   USING process

src_of_abstract_inputs ::=   abstract_sensor
                           | informata
                           | process_outputs

dest_of_abstract_outputs ::=   abstract_actuator
                             | informata
                             | process_input
```

APPENDIX 2

MENUS IN THE ROBOT SHELL

Main Menu for Robot Shell

The screen below is the start-up screen of the Robot Shell. The kinema knowledge generator and the robot simulation managers are integrated under the robot shell. This screen allows the description of kinema instances and knowledge generation under the workcell description choice of the menu. Code generation for forward solution modules are permitted under this menu for the manipulators modelled. The robot system symbol manipulator can be accessed directly from the robot shell to perform symbolic computations on scalars, vectors and matrices. The single and two-robot simulation managers can be accessed through the simulator option on the robot shell initial menu. The menu hierarchy of the Robot Shell is shown in Figure A.1.

```
+---------------------------------------------------------------+
|                 ROBOT SHELL - Version 1.0                     |
|                                                               |
| 1) WORKCELL DESCRIPTION                                       |
|    To describe primitive models and generate knowledge for    |
|    instances                                                  |
|                                                               |
| 2) TASK DESCRIPTION                                           |
|    To describe tasks for defined robot workcells              |
|                                                               |
| 3) CODE GENERATION                                            |
|    To generate code for simulation and control from derived   |
|    knowledge                                                  |
|                                                               |
| 4) SYMBOL MANIPULATOR                                         |
|    Direct access to the robot shell symbol manipulator        |
|                                                               |
| 5) SIMULATOR                                                  |
|    To perform workcell and task simulations for off-line      |
|    testing                                                    |
|                                                               |
| 6) CIM NETWORK INTERFACE                                      |
|    For direct access to the actual manufacturing workcell     |
|                                                               |
| Q) EXIT TO SYSTEM                                             |
+---------------------------------------------------------------+
```

Workcell Description Menu

Under the Workcell Description menu of the robot shell, kinema object instances can be specified and derived knowledge about these kinema instances can be generated. After MEDLI is developed, primitive object models can be described from this menu using MEDL. This selection would permit aggregations to be specified using MEDL and control routines also to be interfaced to the knowledge generators.

```
+-----------------------------------------------------------------+
|                     WORKCELL DESCRIPTION                        |
|                                                                 |
| 1) DEFINE PRIMITIVE OBJECT MODELS                               |
|    To specify object models for driving knowledge generation   |
|                                                                 |
| 2) GENERATE INSTANCES FOR PRIMITIVES                            |
|    Generate derived knowledge for object instances              |
|                                                                 |
| Q) EXIT TO ROBOT SHELL                                          |
|                                                                 |
+-----------------------------------------------------------------+
```

Instance Generation Menu

The Instance Generation menu controls derived knowledge generation for basis and aggregation objects. Kinema is a MEDL described aggregation for which knowledge generators have been implemented. As the system develops, the knowledge generators for the basis, system and user defined aggregations will be accessible through this menu.

```
+------------------------------------------------------------------+
|                  GENERATE INSTANCES FOR PRIMITIVES                |
|                                                                    |
| 1) KINEMA INSTANTIATION                                            |
|    Generate derived knowledge for kinema instances                 |
|                                                                    |
| 2) SENSOR INSTANTIATION                                            |
|    Generate derived knowledge for sensor instances                 |
|                                                                    |
| 3) STATICA INSTANTIATION                                           |
|    Generate derived knowledge for statica instances                |
|                                                                    |
| 4) PROCESS INSTANTIATION                                           |
|    Generate derived knowledge for process instances                |
|                                                                    |
| 5) INFORMATA INSTANTIATION                                         |
|    Generate derived knowledge for informata instances              |
|                                                                    |
| 6) WORKCELL INTERCONNECTION DESCRIPTION                            |
|    Specify the interconnection of the objects in the workcell      |
|                                                                    |
| Q) EXIT TO WORKCELL DESCRIPTION                                    |
|                                                                    |
+------------------------------------------------------------------+
```

Kinema Instantiation Menu for Robot Manipulators

Under the Kinema Instantiation menu, Kinema instances can be specified and their symbolic derived knowledge generated using the kinema derived knowledge generation programs. In our example manipulator knowledge generators have been developed. Choices should be available for describing new kinema description, editing and modifying existing descriptions, or knowledge generation for the current instances. From the Denavit and Hartenberg (D-H) notational specification of manipulators, their homogeneous A matrices can be generated. The A matrices multiplied in order gives the symbolic forward solution for the manipulator. Forward solution of a manipulator describes its end-effector pose relative to its base frame description. For the reverse solution, A-matrix inverses and their multiplicative terms can be generated from this menu. Symbolic end-frame and mid-frame Jacobians can be computed from this menu as well. These Jacobian matrices are useful for manipulator control and analysis of robot manipulator singularities.

```
+----------------------------------------------------------------+
|                        KINEMA INSTANTIATION                    |
|                                                                |
| 1) SPECIFY KINEMA                                              |
|    Make specification catalog entries for kinema instance      |
|                                                                |
| 2) GENERATE A MATRICES                                         |
|    Generate A matrices into the derived knowledge base         |
|                                                                |
| 3) FORWARD & REVERSE SOLUTION                                  |
|    Compute forward and reverse solutions for kinema instances  |
|                                                                |
| 4) END & MID FRAME JACOBIANS                                   |
|    Compute end-frame and mid-frame Jacobians for kinema        |
|    instances                                                   |
|                                                                |
| 5) SAVE ROBOT SYMBOLIC COMPUTATIONS                            |
|    Make permanent copy of derived knowledge in database        |
|                                                                |
| 6) CHANGE DEFAULT ROBOT BEING MODELLED                         |
|    Change the kinema instance being modelled                   |
|                                                                |
| Q) EXIT TO GENERATE OTHER PRIMITIVE INSTANCES                  |
+----------------------------------------------------------------+
```

Manipulator Parameter Collection Screen

The Parameter Collection Screen shown below permits user specification catalog data entry for kinema aggregation instances. D-H notational tables are used as the screen format for kinema descriptions. This information is sufficient for kinematic modelling of robot manipulators.

```
+-----------------------------------------------------------------+
|                  PARAMETER COLLECTION SCREEN                    |
|      +---------------------------------------------------------+|
|      |     |     |        |     |     |        |       |       ||
|      | DOF | VAR | Theta  |  d  |  a  | Alpha  |  Max  |  Min  ||
|      |     |     |        |     |     |        |       |       ||
|      ----------------------------------------------------------- |
|      |  1  |  r  |  90.0  | d1  | 0   |  90.0  |  0.0  |  0.0  ||
|      |  2  |  r  |   0.0  | 0   | a2  |   0.0  |  0.0  |  0.0  ||
|      |  3  |  r  |  90.0  | d3  | 0   |  90.0  |  0.0  |  0.0  ||
|      |  4  |  r  | -90.0  | d4  | 0   | -90.0  |  0.0  |  0.0  ||
|      |  5  |  r  |  90.0  | 0   | 0   |  90.0  |  0.0  |  0.0  ||
|      |  6  |  r  |   0.0  | 0   | 0   |   0.0  |  0.0  |  0.0  ||
|      |     |     |        |     |     |        |       |       ||
|      |     |     |        |     |     |        |       |       ||
|      |     |     |        |     |     |        |       |       ||
|      +---------------------------------------------------------+|
|                                                                 |
|   <CTRLD> to delete last DOF and <CTRLI> to insert a DOF        |
|   <- , -> , ^ , v : for Left, Right, Up, Down                   |
|   <CTRLE> to exit, <CTRLC> to quit without update               |
|                                                                 |
+-----------------------------------------------------------------+
```

<u>Manipulator Forward and Reverse Solution Menu</u>

The Symbolic Forward & Reverse Solution menu provides the options of generating the multiplication of all the A matrices to compute the forward solution for a modelled manipulator. The A matrix inversions and multiplications of the inverse A matrices can also be generated under this menu. The inverse matrices aid in the computation of reverse solutions for manipulators. The reverse solution program also identifies terms from these products to be used for solving the reverse solutions for the various degrees-of-freedom of the manipulators. When general reverse solutions techniques become available, they will be inserted into the

```
+------------------------------------------------------------------+
|                SYMBOLIC FORWARD & REVERSE SOLUTION               |
|                                                                  |
| 1 - FORWARD SOLUTION FORMULATION                                 |
|      Form the Tn matrix by multiplying the A1*A2*...*An          |
|                                                                  |
| 2 - REVERSE SOLUTION FORMULATION                                 |
|      Form the matrices that will aid the reverse solution        |
|      formulation A1inv, A2inv,...,Aninv and the product terms    |
|      of the inverses                                             |
|                                                                  |
| 3 - DIRECTORY OF SYMBOLIC MATRICES                               |
|      Show a list of all Symbolic Matrices in the database        |
|                                                                  |
| 4 - DISPLAY SYMBOLIC MATRICES                                    |
|      Display Data in the Symbolic Matrices in The Database       |
|                                                                  |
| 5 - PRINT SYMBOLIC MATRICES                                      |
|      Print Data in the Symbolic Matrices in the Database         |
|                                                                  |
| Q - to QUIT                                                      |
+------------------------------------------------------------------+
```

Manipulator Jacobian Generation Menu

Under this robot shell menu, symbolic end-frame and mid-frame Jacobians can be computed for robot manipulators. The end-frame computation depends on the number of degrees of freedom of the manipulator. The mid-frame for the Jacobian computations can be chosen arbitrarily by the user to lie between the origin and the end-frame. Symbolic Jacobians are (DOF X 6) matrices and can be used for studying the positional and velocity singularities in manipulator workspaces.

```
+------------------------------------------------------------+
|              GENERATE AND PRINT JACOBIANS                   |
|                                                             |
| 1) END FRAME JACOBIAN                                       |
|    Generate the Symbolic End Frame Jacobian for the         |
|    Manipulator                                              |
|                                                             |
| 2) MID FRAME JACOBIAN                                       |
|    Generate the Symbolic Mid Frame Jacobian for the         |
|    Manipulator at the User Specified Mid Frame              |
|                                                             |
| 3) DISPLAY JACOBIAN                                         |
|    Display the Jacobian from the Symbolic Database          |
|                                                             |
| 4) PRINT JACOBIAN                                           |
|    Print the Jacobian from the Symbolic Database            |
|                                                             |
| 5) DISPLAY JACOBIAN NAMES                                   |
|    Display the names of all the jacobians in the symbolic   |
|    database                                                 |
|                                                             |
| q, Q) QUIT                                                  |
+------------------------------------------------------------+
```

<u>Robot Shell Symbol Manipulator Menu</u>

The Symbol Manipulation menu gives users a direct access to the robot shell symbol manipulator program. In this program, scalars are arbitrary length strings, vectors are (4 X 1) arrays of scalars and matrices are (4 X 4) arrays of scalars. Various monodic and dyadic operations are permitted between the different named instances of scalars, vectors and matrices. The symbol manipulator subroutines are available in separately compiled modules and were used extensively in the development of the various symbol manipulation tools in the robot shell. The symbol manipulator also provides a command interpreter which parses and executes symbolic computations on data stored in symbolic knowledge bases. Additional information about the symbol manipulator can be found in the symbol manipulator user's manual.

```
+------------------------------------------------------------------+
|                  SYMBOL MANIPULATION FOR ROBOTICS                |
|                                                                  |
| Symbolic operations are supported on matrices, vectors and       |
| scalars. Monodic and Diadic operations are allowed. The          |
| operators can be the same or different types. Refer              |
| instruction manual for legal symbolic operations                 |
|                                                                  |
| A) COLLECT DATA                                                  |
|      To define and store scalar, vector or matrix data           |
|                                                                  |
| B) PERFORM COMPUTATIONS                                          |
|      Perform diadic or monodic operations on scalars, vectors    |
|      or matrices                                                 |
|                                                                  |
| C) SHOW FILE DATA                                                |
|      Show stored data and results from computations              |
|                                                                  |
| Q) To quit                                                       |
|                                                                  |
+------------------------------------------------------------------+
```

Robot Shell Simulation Manager

The single and two-robot Simulation Manager can be accessed through the Robot Simulations menu of the robot shell. Both these simulators currently use Unimation Puma-260 manipulators.

```
+---------------------------------------------------------------+
|                                                               |
|                        ROBOT SIMULATIONS                      |
|                                                               |
| Simulation Manager for Robots. Performs Simulations of Forward|
| and Reverse Solutions of the Following Robots. Also generates |
| Joint Interpolated, Paul's Cartesian Interpolated and Taylor's|
| Cartesian Trajectories.                                       |
|                                                               |
| 1) SIMULATE PUMA-260                                          |
|                                                               |
|       To Simulate a Unimation Puma-260                        |
|                                                               |
| 2) SIMULATE TWO ROBOTS                                        |
|                                                               |
|       To Simulate a two robot workcell with two puma-260's    |
|                                                               |
| Q) EXIT TO ROBOT SHELL                                        |
|                                                               |
+---------------------------------------------------------------+
```

Single Robot Simulation Manager Menu

In the single robot simulation manager, users can perform numerical forward and reverse solutions. The entire solution tree will be displayed on the computation of the reverse solution. Motion planning for Puma-260's using joint interpolated schemes and cartesian interpolated schemes of trajectory planning are permitted. Dynamics computations for the Puma-260 are also allowed. The user is given the choice of tailoring the simulation environment and results to their tastes and requirements.

```
+------------------------------------------------------------------+
|                  SIMULATION MANAGER FOR PUMA-260                 |
|                                                                  |
|  1. Forward Solution                6. Toggle Deg/Rad display    |
|                                                                  |
|  2. Reverse Solution                7. Toggle Pose/Config input  |
|                                                                  |
|  3. Forward into Reverse Solution   8. Change FuzzFactor         |
|                                                                  |
|  4. Collect Parameters              9. Generate Trajectories     |
|                                                                  |
|  5. Pose of solution in sol set     a. Dynamics Simulations      |
|                                                                  |
|                                     b. Change Plot Parameters   |
|                                                                  |
|                        <CTRL>E - EXIT                            |
|------------------------------------------------------------------|
|   <DEGREES>     <CONFIGURATIONS>    FUZZFACTOR : 1e-06           |
|                                                                  |
+------------------------------------------------------------------+
```

<u>Single Robot Trajectory Generation Menu</u>

Under the Trajectory Generation menu, users can generate joint interpolated and Richard Paul's and Taylor's cartesian interpolated trajectories for the specified Puma-260 robot trajectories. Hardcopies can be generated for the six joint variables of the Puma for specified sampling rates. The joint variables can also be plotted on a 5620 DMD screen.

```
+----------------------------------------------------------------+
|              TRAJECTORY GENERATION FOR A PUMA-260              |
|                                                                |
|   1. Joint Interpolation                                       |
|       Vary the configuration angles of the robot by           |
|       interpolating the joint angles to vary smoothly between |
|       each subsequent configuration                            |
|                                                                |
|   2. Cartesian Interpolation - Paul's Scheme                   |
|       Smooth interpolation of the robot configuration using   |
|       Richard Paul's Cartesian Interpolation scheme            |
|                                                                |
|   3. Cartesian Interpolation - Taylor's Scheme                 |
|       Smooth Interpolation of the robot configuration using   |
|       Taylor's Cartesian Interpolation Scheme                  |
|                                                                |
|   4. Toggle PRINT/NO PRINT                                     |
|                                                                |
|   5. Toggle PLOT/NO PLOT                                       |
|                                                                |
|                              <CTRLE> EXIT                      |
+----------------------------------------------------------------+
|    NO PRINT             NO PLOT                        >>>     |
+----------------------------------------------------------------+
```

<u>Single Robot Dynamics Computation Menu</u>

The Dynamics Simulations menu allows Newton-Euler dynamics and Lagrangian dynamics computations and display for a Puma-260 manipulator executing trajectories computed using Richard Paul's scheme. From this menu also, hardcopies and plots for the various degrees-of-freedom of the manipulator can be generated.

```
+-------------------------------------------------------------------+
|              DYNAMICS SIMULATIONS FOR A PUMA-260                  |
|                                                                   |
|   1. Newton-Euler Dynamics : Paul's Trajectory                    |
|       Newton-Euler Dynamics simulation Applied to Paul's          |
|       Cartesian Interpolation Trajectory Generation               |
|                                                                   |
|   2. Lagrangian Dynamics : Paul's Trajectory                      |
|       Lagrangian Dynamics Simulation Applied to Paul's            |
|       Cartesian Interpolation Trajectory Generation               |
|                                                                   |
|   3. Toggle PRINT/NO PRINT                                        |
|                                                                   |
|   4. Toggle PLOT/NO PLOT                                          |
|                                                                   |
|                              <CTRLE> EXIT                         |
+-------------------------------------------------------------------+
|    NO PRINT            NO PLOT                        >>>>        |
|                                                                   |
+-------------------------------------------------------------------+
```

Two Robot Simulation Manager

The two-robot simulation manager is modelled quite similar to the one-robot simulation manager. Static manipulation and kinetic manipulation motion tasking primitives can be simulated and tested using two Puma-260 manipulators. Workcell configurations and grasp transformations can be changed for the two robots while simulating the two-robot motion primitives.

```
+----------------------------------------------------------------+
|                SIMULATION MANAGER FOR TWO PUMAS                |
|                --------------------------------                |
|                                                                |
| 1. Static Manipulation            5. Toggle Deg/Rad display    |
|                                                                |
| 2. Kinetic Manipulation           6. Toggle Pose/Config input  |
|                                                                |
| 3. Collect Parameters             7. Toggle Slow/High Speed    |
|                                                                |
| 4. Change Plot Characteristics    8. Toggle Plot/No Plot       |
|                                                                |
|                                   9. Change FuzzFactor         |
|                                                                |
|                      <CTRL>E - EXIT                            |
+----------------------------------------------------------------+
| <DEGREES>  <POSES>  <SLOW>  <PLOT>  FUZZFACTOR : 0.000001      |
|                                                                |
+----------------------------------------------------------------+
```

<u>Robot Primitives Parameter Collection Menu</u>

Using the parameter specification menu, the workcell configurations of the two robot workcell can be altered by changing uB1 and uB2 which specify the robot base frame relative to the workcell universal frame. From this menu, the grasp transformations for the two manipulators, the initial configurations of the manipulators, the rendezvous pose for static manipulation and the rendezvous trajectory for the kinetic manipulation can be changed and tested. The code used in this simulation manager can be used later for testing the task plans generated by a task planner and grasping transformations computed for cooperating robot tasks.

```
+--------------------------------------------------------------+
|            PARAMETER COLLECTION SCREEN FOR TWO PUMAS         |
|            ----------------------------------------          |
|                                                              |
| 1. Change uB1                  7. Collect Static Manip Params |
|                                                              |
| 2. Change uB2                  8. Collect Kinetic Manip Params|
|                                                              |
| 3. Change cmG1                 9. Toggle Deg/Rad Display     |
|                                                              |
| 4. Change cmG2                 a. Toggle Pose/Config input   |
|                                                              |
| 5. Change Robot 1 Init Config                                |
|                                                              |
| 6. Change Robot 2 Init Config                                |
|                                                              |
|                             <CTRL>E - EXIT                   |
+--------------------------------------------------------------+
|  <DEGREES>    <POSES>                FUZZFACTOR : 0.000001    |
|                                                              |
+--------------------------------------------------------------+
```

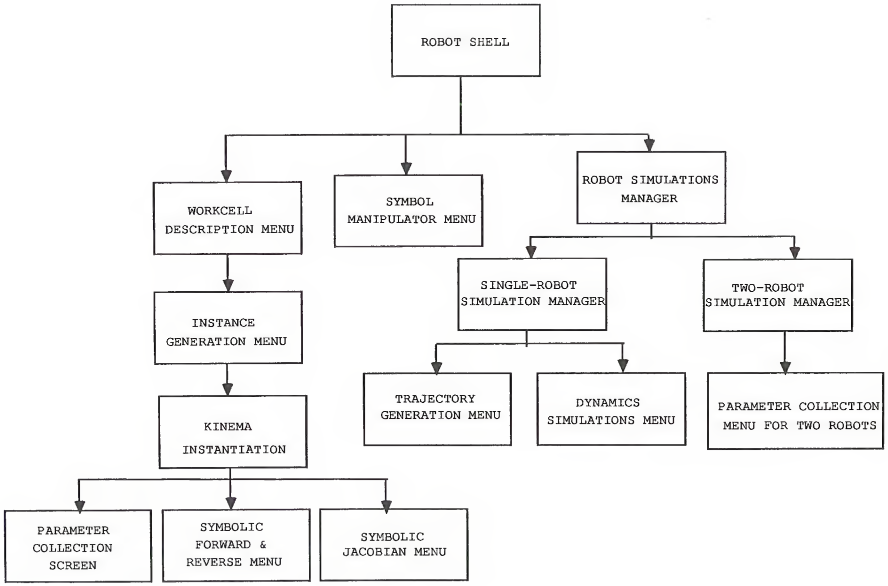

ROBOT SHELL
WORKCELL DESCRIPTION MENU
SYMBOL MANIPULATOR MENU
ROBOT SIMULATIONS MANAGER
INSTANCE GENERATION MENU
SINGLE-ROBOT SIMULATION MANAGER
TWO-ROBOT SIMULATION MANAGER
KINEMA INSTANTIATION
TRAJECTORY GENERATION MENU
DYNAMICS SIMULATIONS MENU
PARAMETER COLLECTION MENU FOR TWO ROBOTS
PARAMETER COLLECTION SCREEN
SYMBOLIC FORWARD & REVERSE MENU
SYMBOLIC JACOBIAN MENU

FIGURE A.1 Menu Hierarchy in the Robot Shell

REFERENCES

Acker, F.E., Ince, I.A., and Ottinger, B.D., "TROIKABOT--A Multi-armed Assembly-Robot," Proceedings Robots 9 Conference, Detroit, MI, 1985.

Bogen, R. A., MACSYMA Reference Manual, Version 6, Laboratory for Computer Science, Cambridge, MA, 1977.

Chand, S., "Cooperation and Coordination of Computer Controlled Manipulators," PhD Dissertation, University of Florida, Gainesville, FL, December 1984.

Chand, S., and Doty, K.L., "Trajectory-Specification and Load Distribution for Closed-Loop Multi-Manipulator Systems," Proceedings of SOUTHEASTCON 84, Tampa, FL, March 1984.

Crane, C.D., "Motion Planning and Control of Robot Manipulators via Application of a Computer Graphics Animated Display," PhD Dissertation, University of Florida, Gainesville, FL, 1987.

Doty, K.L., "Cooperating Robots," Report DAAE07-83-MR011, US Army Tank-Automotive Command, Gainesville, FL, September 1983.

Faverjon, B., "Object Level Programming of Industrial Robots," Proceedings of the IEEE International Conference on Robotics, Raleigh, NC, March 1987.

Finkel, R., Taylor, R., Bolles, R., Paul, R., and Feldman, J., "AL, A Programming System for Automation," Stanford Artificial Intelligence Laboratory Memo AIM-243, Stanford, CA, November 1974.

Gavrilovic, M.M., and Selic, B.V., "Synergistic Control of Bilateral Manipulator Systems," Proceedings of the 4th International Symposium on Industrial Robots, Tokyo, Japan, November 1974.

Govindaraj, S., "Tasking Primitives for Cooperating Multi-Robot Workcells," Machine Intelligence Laboratory, University of Florida, MIL report MIL0687SG1, Gainesville, FL, June 1987.a

Govindaraj, S., "A Modelling Basis for Multi-Robot Workcells," Machine Intelligence Laboratory, University of Florida, MIL report MIL0687SG2, Gainesville, FL, June 1987.b

Govindaraj, S., "Puma-260 Simulation Manager User's Guide," Machine Intelligence Laboratory, University of Florida, MIL report MIL0687SG3, Gainesville, FL, June 1987.c

Govindaraj, S., and Doty, K.L., "Considerations in Robot Cooperation," Machine Intelligence Laboratory, University of Florida, MIL report MIL0885SG1, Gainesville, FL, August 1985.

Govindaraj, S., and Doty, K. L., "General Purpose Robot System and Task Development Facility," Proceedings of Robots 10 Conference, Chicago, IL, April 1986.

Hayward, V., and Paul, R. P., "Robot Manipulator Control Under UNIX RCCL: A Robot Control "C" Library," The International Journal of Robotics Research, Vol. 5, No. 4, pp. 94-111, Winter 1986.

Hornick, M.L., and Ravani, B., "Computer-Aided Off-Line Planning and Programming of Robot Motion," The International Journal of Robotics Research, Vol. 4, No. 4, pp. 18-31, Winter 1986.

Jacobsen, S., Wood, J., Knutti, D.F., and Biggers, K.B., "The Utah/M.I.T. Dextrous Hand: Work in Progress," The International Journal of Robotics Research, Vol. 3, No. 4, pp. 21-50, Winter 1984.

Kemper, A., and Wallrath, M., "An Analysis of Geometric Modelling in Database Systems," ACM Computing Surveys, Vol. 19, No. 1, pp 47-91, March 1987.

Kilmer, W.S., McCain, H.G., Juberts, M., and Legowik, S.A., "Watchdog Safety Computer Design and Implementation," Proceedings of Robots 8 Conference, Detroit, MI, June 1984.

Kirshbrown, R. H., and Dorf, R. C., "KARMA--A Knowledge-Based Robot Manipulation System: Determining Problem Characteristics," Proceedings of Robots-8, Detroit, MI, June 1984.

Lieberman, L., and Wesley, M., "Autopass: An Automatic Programming System for Computer Controlled Mechanical Assembly," IBM Journal of Research and Development, Vol. 21, No. 4, pp. 321-333, July 1977.

Luh, J.Y.S., and Zheng, Y.F., "Constrained Relations between Two Coordinated Industrial Robots for Motion Control," The International Journal of Robotics Research, Vol. 6, No. 3, pp. 60-70, Fall 1987.

Malm, J. F., "Symbolic Matrix Multiplication with LISP Programming," Proceedings Robots-8, Detroit, MI, June 1984.

Manseur, R., and Doty, K.L., "An Inverse Kinematic Algorithm for Solving 6-DOF-Manipulators: Discovery of 16 Real Solution Sets," Submitted for publication, June 1987.

McGhee, R.B., "Vehicular Legged Locomotion," Advances in Automation and Robotics, ed. Saridis, G.N., Greenwich, CT, Jai Press, 1984.

Mogal, J. S., "IGRIP-A Graphics Simulation Program for Workcell Layout and Off-Line Programming," Proceedings of Robots 10, Chicago, IL, April 1986.

Nakamura, Y., Nagai, K., and Yoshikawa, T., "Mechanics of Coordinative Manipulation by Multiple Robotic Mechanisms," Submitted for Publication, March 1987.

Nakano, E., Ozaki, S., Ishida, T., and Kato, I., "Cooperational Control of the Anthropomorphous Manipulator "MELARM"," Proceedings of the 4th International Symposium on Industrial Robots, Tokyo, Japan, November 1974.

NSF Workshop on Coordinated Robots, San Diego, CA, January 1987.

Orin, D. E., "Interactive Control of a Six-legged Vehicle with Optimization of both Stability and Energy," PhD dissertation, The Ohio State University, Columbus, OH, 1976.

Orin, D.E. and Oh, S.Y., "Control of Force Distribution in Robotic Mechanisms Containing Closed Kinematic Chains," Transactions of the ASME, Vol. 102, pp. 134-141, 1981.

Pal, S., and Doty, K.L., "Hierarchical Task Planning with Application to Dual Robot Workcells," Proceedings of Conference on AI in Manufacturing, Long Beach, CA, October 1987.

Paul, R. C., "Manipulator Cartesian Path Control," IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-9, pp. 702-711, 1979.

Paul, R. P., Robot Manipulators: Mathematics, Programming, and Control, The MIT Press, Cambridge, MA, 1981.

Pavelle, R., Rothstein, M., and Fitch, J., "Computer Algebra," Scientific American, pp. 136-152, December 1981.

Raibert, M.H., Brown, H.B., and Chepponis, M., "Experiments in Balance with a 3D one-legged Hopping Machine," The International Journal of Robotics Research, Vol. 3, No. 2, pp. 75-92, Summer 1984.

Rembold, U., and Blume, C., "Programming Languages And Systems For Assembly Robots," Computers in Mechanical Engineering, Vol. 2, No. 4, pp. 61-68, January 1984.

Ruoff, C., "TEACH--A Concurrent Robot Control Language," Proceedings of 1979 IEEE Computer Software Applications Conference, Chicago, IL, November 1979.

Salisbury, J.K., and Craig, J.J., "Articulated Hands: Force Control and Kinematic Issues," The International Journal of Robotics Research, Vol. 1, No. 1, pp. 4-17, Spring 1982.

Simpson, J., Hocken, R., and Albus, J., "The Automated Manufacturing Research Facility of the National Bureau of Standards," Journal of Manufacturing Systems, Vol. 1, No. 1., pp. 17-32, January 1982.

Stauffer, R.N., "Two-arm Robot Uses one Control System and Power Unit to Cut Cost and Simplify Operation," Industrial Robots, Volume 2: Applications, ed. by Tanner, W.R., Published by Robotics International of SME, Dearborn, MI, 1981.

Su, S.Y.W., "Modelling Integrated Manufacturing Data with SAM*," IEEE Computer, pp. 34-49, January 1986.

Su, S.Y.W., Krishnamurthy, V., and Lam, H., "An Object-oriented Semantic Association Model (OSAM*)," To be published in AI in Industrial Engineering and Manufacturing: Theoretical Issues and Applications, ed. by Kumara, S., Kashyap, R.L., and Soyster, A.L., Published by American Institute of Industrial Engineers, 1988.

Su, S.Y.W., and Raschid, L., "Incorporating Knowledge Rules in a Semantic Data Model: An Approach to Integrated Knowledge Management, " A. I. Applications Conference, Miami, FL, December 1985.

Tarn, T.J., Bejczy, A.K., and Yun, X., "Coordinated Control of Two Arm Robots," Proceedings of 1986 IEEE International Conference on Robotics and Automation, San Francisco, April 1986.

Taylor, R.H., "Planning and Execution of Straight Line Manipulator Trajectories," IBM Journal of Research and Development, Vol. 23, pp. 424-436, 1979.

Zheng, Y.F., Luh, J.Y.S., and Jia, P.F., "A Real-Time Distributed Computer System for Coordinated-Motion Control of Two Industrial Robots," Proceedings of 1987 IEEE International Conference on Robotics and Automation, Raleigh, NC, April 1987.

BIOGRAPHICAL SKETCH

Subbian Govindaraj was born in Coimbatore, a small Indian town in Tamilnadu, on December 5th, 1957. He went to Mani High School, known throughout the district of Coimbatore for its excellence in education. He graduated as the top ranked student from the P.S.G. College of Technology in the University of Madras with a Bachelor of Engineering (Honours) degree in May 1980, majoring in electrical and electronics engineering.

He was awarded the Master of Engineering in electrical engineering by the University of Florida in December 1982 and will receive the degree of Doctor of Philosophy in electrical engineering in April 1988. He is very happily married to Nalini Mulpuri and plans to pursue a career in research, development and teaching.

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

[illegible]

Keith L. Doty, Chairman
Professor of Electrical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Giuseppe Basile

Giuseppe Basile
Professor of Electrical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Herman Lam

Herman Lam
Associate Professor of Electrical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

[illegible]

Robert L. Sullivan
Professor of Electrical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

[illegible]

Yuan H. Chow
Associate Professor of Computer and Information Sciences

This dissertation was submitted to the Graduate Faculty of the College of Engineering and to the Graduate School and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

April 1988

[illegible]

Dean, College of Engineering

Dean, Graduate School

72-1
D-745
#578

15